

EDITORS

**M. HUBA, S. SKOGESTAD, M. FIKAR, M. HOVD,  
T.A. JOHANSEN, B. ROHAL'ILKIV**

**SELECTED TOPICS ON  
CONSTRAINED  
AND NONLINEAR CONTROL**  
WORKBOOK



2011

**STU BRATISLAVA and NTNU TRONDHEIM**

M. Huba, S. Skogestad, M. Fikar, M. Hovd,  
T. A. Johansen, B. Rohal'-Ilkiv

Editors

**Selected Topics on  
Constrained and Nonlinear  
Control  
Workbook**

STU Bratislava – NTNU Trondheim

Copyright © 2011 authors  
Compilation: Miroslav Fikar  
Cover: Tatiana Hubová  
Printed and bounded in Slovakia by Miloslav Roubal ROSA, Dolný Kubín  
and Tlačiareň Vrábek, Dolný Kubín  
ISBN: 978-80-968627-3-3

# Preface

This workbook was created within the NIL-I-007-d Project “Enhancing NO-SK Cooperation in Automatic Control” (ECAC) carried out in 2009-2011 by university teams from the Slovak University of Technology in Bratislava and from the Norwegian University of Science and Technology in Trondheim. As it is already given by the project title, its primary aim was enhancing cooperation in academic research in the automatic control area in the partner institutions. This was achieved by supporting broad spectrum of activities ranging from student mobilities at the MSc. and PhD. level, staff mobilities, organization of multilateral workshop and conferences, joint development of teaching materials and publishing scientific publications. With respect to the original project proposal, the period for carrying out the foreseen activities was reasonably shortened and that made management of the all work much more demanding. Despite of this, the project has reached practically all planned outputs – this workbook represents one of them – and we believe that it really contributes to the enhancement of Slovak-Norwegian cooperation and to improvement of the educational framework at both participating universities. Thereby, I would like to thank all colleagues participating in the project activities at both participating universities and especially professor Sigurd Skogestad, coordinator of activities at the NTNU Trondheim, associate professor Katarína Žáková for managing the project activities, and professor Miroslav Fikar for the patient and voluminous work with collecting all contribution and compilation of all main three project publications (textbook, workbook, and workshop preprints).

Bratislava  
2.1.2011

*Mikuláš Huba*  
*Project coordinator*



## Acknowledgements

The authors and editors are pleased to acknowledge the financial support the grant No. NIL-I-007-d from Iceland, Liechtenstein and Norway through the EEA Financial Mechanism and the Norwegian Financial Mechanism. This book is also co-financed from the state budget of the Slovak Republic.



# Contents

<b>1</b>	<b>Problems in Anti-Windup and Controller Performance Monitoring</b>	<b>1</b>
	Morten Hovd and Selvanathan Sivalingam	
1.1	Anti-Windup: Control of a Distillation Column with Input Constraints	1
1.1.1	Notation	2
1.1.2	Some Background Material on Anti-windup	2
1.1.3	Decoupling and Input Constraints	9
1.1.4	The Plant Model used in the Assignment	10
1.1.5	Assignment	11
1.1.6	Solution	12
1.1.7	Simulations	15
1.1.8	Anti-windup	19
1.1.9	Matlab Code	22
1.2	Anti-windup with PI Controllers and Selectors	27
1.2.1	Introduction to Selective Control	27
1.2.2	The Control Problem in the Assignment	29
1.2.3	Assignment	30
1.2.4	Solution	30
1.3	Stiction Detection	34
1.3.1	Assignment	35
1.3.2	Solution	36
1.3.3	Conclusions	44
1.4	Controller Performance Monitoring using the Harris Index	45
1.4.1	Assignment	46
1.4.2	Solution	46
	References	48



<b>2</b>	<b>Optimal Use of Measurements for Control, Optimization and Estimation using the Loss Method: Summary of Existing Results and Some New</b> .....	53
	Sigurd Skogestad and Ramprasad Yelchuru and Johannes Jäschke	
2.1	Introduction .....	53
2.2	Problem Formulation .....	54
2.2.1	Classification of Variables .....	54
2.2.2	Cost Function .....	54
2.2.3	Measurement Model .....	55
2.2.4	Assumptions .....	55
2.2.5	Expected Set of Disturbances and Noise .....	55
2.2.6	Problem .....	56
2.2.7	Examples of this Problem .....	56
2.2.8	Comments on the Problem .....	56
2.3	Solution to Problem: Preliminaries .....	57
2.3.1	Expression for $\mathbf{u}_{opt}(\mathbf{d})$ .....	57
2.3.2	Expression for $\mathbf{J}$ around $\mathbf{u}_{opt}(\mathbf{d})$ .....	57
2.3.3	Expression for $\mathbf{J}_u$ around Moving $\mathbf{u}_{opt}(\mathbf{d})$ .....	58
2.3.4	Optimal Sensitivities .....	58
2.4	The Loss Method .....	59
2.4.1	The Loss Variable $\mathbf{z}$ as a Function of Disturbances and Noise .....	59
2.4.2	Loss for Given $\mathbf{H}$ , Disturbance and Noise (Analysis) .....	59
2.4.3	Worst-case and Average Loss for Given $\mathbf{H}$ (Analysis) .....	60
2.4.4	Loss Method for Finding Optimal $\mathbf{H}$ .....	60
2.5	Reformulation of Loss Method to Convex Problem and Explicit Solution .....	63
2.6	Structural Constraints on $\mathbf{H}$ .....	65
2.7	Some Special Cases: Nullspace Method and Maximum Gain Rule .....	66
2.7.1	No Measurement Noise: Nullspace Method (“full $\mathbf{H}$ ”) .....	67
2.7.2	No Disturbances .....	68
2.7.3	An Approximate Analysis Method for the General Case: “Maximum Gain Rule” .....	68
2.8	Indirect Control and Estimation of Primary Variable .....	70
2.8.1	Indirect Control of $\mathbf{y}_1$ .....	71
2.8.2	Indirect Control of $\mathbf{y}_1$ Based on Estimator .....	71
2.9	Estimator for $\mathbf{y}_1$ Based on Data .....	72
2.9.1	Data Approach 1 .....	72
2.9.2	Data Approach 2: Loss Regression .....	72
2.9.3	Modification: Smoothing of Data .....	75
2.9.4	Numerical Tests .....	76
2.9.5	Test 1. Gluten Test Example from Harald Martens .....	76
2.9.6	Test 2. Wheat Test Example from Bjorn Alsberg (Kalivas, 1997) .....	77

2.9.7	Test 3. Our Own Example	78
2.9.8	Comparison with Normal Least Squares	79
2.10	Discussion	80
2.10.1	Gradient Information	80
2.10.2	Relationship to NCO tracking	80
2.11	Appendix	81
	References	86
<b>3</b>	<b>Measurement polynomials as controlled variables – Exercises</b>	<b>91</b>
	Johannes Jäschke and Sigurd Skogestad	
3.1	Introduction	91
3.2	Simple exercise	91
3.3	Isothermal CSTR Case Study	92
3.4	Solution	93
3.4.1	Component Balance	93
3.4.2	Optimization Problem	94
3.4.3	Optimality Conditions	94
3.4.4	Eliminating Unknowns $k_1, k_2$ and $c_B$	95
3.4.5	The Determinant	95
3.4.6	Maple Code	96
<b>4</b>	<b>Multi-Parametric Toolbox</b>	<b>101</b>
	Michal Kvasnica	
4.1	Multi-Parametric Toolbox	101
4.1.1	Download and Installation	102
4.2	Computational Geometry in MPT	103
4.2.1	Polytopes	103
4.2.2	Polytope Arrays	106
4.2.3	Operations on Polytopes	107
4.2.4	Functions Overview	113
4.3	Exercises	115
4.4	Solutions	122
4.5	Model Predictive Control in MPT	131
4.5.1	Basic Usage	133
4.5.2	Closed-loop Simulations	134
4.5.3	Code Generation and Deployment	135
4.5.4	Advanced MPC using MPT and YALMIP	136
4.5.5	Analysis	141
4.5.6	System Structure <code>sysStruct</code>	147
4.5.7	Problem Structure <code>probStruct</code>	151
4.6	Exercises	156
4.7	Solutions	162
	References	166

<b>5</b>	<b>Implementation of MPC Techniques to Real Mechatronic Systems</b> .....	171
	Gergely Takács and Tomáš Polóni and Boris Rohal'-Ilkiv and Peter Šimončič and Marek Honek and Matúš Kopačka and Jozef Csambál and Slavomír Wojnar	
5.1	Introduction .....	172
5.2	MPC Methods for Vibration Control .....	173
5.2.1	Introduction .....	173
5.2.2	Hardware .....	174
5.2.3	Quadratic Programming based MPC .....	177
5.2.4	Newton-Raphson's Suboptimal MPC .....	183
5.2.5	Multi-Parametric MPC .....	194
5.2.6	Conclusion .....	198
5.3	AFR Control .....	200
5.3.1	Introduction .....	200
5.3.2	Hardware Description .....	201
5.3.3	AFR Model Design .....	206
5.3.4	Predictive Control .....	216
5.3.5	Results of a Real-Time Application of a Predictive Control .....	219
5.3.6	Conclusion .....	222
	References .....	222
<b>6</b>	<b>Laboratory Model of Coupled Tanks</b> .....	229
	Vladimír Žilka and Mikuláš Huba	
6.1	Introduction .....	229
6.2	Coupled Tanks – Hydraulic Plant .....	230
6.2.1	Identification .....	232
6.2.2	Sensors Calibration .....	235
6.2.3	Automatic Calibration and Identification .....	236
6.2.4	Some Recommendation for Users .....	239
	References .....	241
<b>7</b>	<b>Constrained PID Control Tasks for Coupled Tanks Control</b> .....	247
	Vladimír Žilka and Mikuláš Huba	
7.1	Introduction .....	247
7.2	Basic P and PI controllers .....	248
7.2.1	PI-controller .....	250
7.2.2	PI <sub>1</sub> controller .....	251
7.3	Linearization around a fixed operating point .....	255
7.4	Exact Feedback Linearization .....	257
7.5	PD <sub>2</sub> controller .....	260
7.6	Conclusion .....	269
	References .....	269

<b>8</b>	<b>Remote Laboratory Software Module for Thermo Optical Plant</b> .....	275
	Pavol Bisták	
8.1	Introduction .....	275
8.2	Technical Requirements .....	276
8.2.1	Server .....	276
8.2.2	Client Computer .....	276
8.3	Installation .....	276
8.3.1	Server Installation .....	276
8.3.2	Client Installation .....	277
8.4	Running the Client Server Application .....	277
8.5	Client User Interface .....	279
8.5.1	Settings .....	280
8.5.2	Server IP Address and Control Buttons .....	281
8.6	Running the Experiment .....	283
8.7	Rules for Creation of Models in Simulink .....	284
8.8	Conclusion .....	286
<b>9</b>	<b>Constrained PID control Tasks for Controlling the Thermo Optical Plant</b> .....	291
	Peter Ťapák and Mikuláš Huba	
9.1	Thermo-optical Plant uDAQ28/LT – Quick Start .....	292
9.1.1	Installation in Windows Operating System .....	292
9.2	Light Channel Control .....	298
9.2.1	Feedforward Control .....	299
9.2.2	$I_0$ Controller .....	302
9.2.3	Filtered Predictive $I_0$ Controller .....	308
9.2.4	$PI_0$ and $FPI_0$ Controllers .....	313
9.2.5	$PI_1$ controller .....	319
9.2.6	Filtered Smith Predictor (FSP) .....	321
	References .....	327



## List of Contributors

Pavol Bisták

Institute of Control and Industrial Informatics, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, e-mail: [pavol.bistak@stuba.sk](mailto:pavol.bistak@stuba.sk)

Jozef Csambál

Institute of Measurement, Automation and Informatics, Faculty of Mechanical Engineering, Slovak University of Technology in Bratislava, e-mail: [jozef.csambal@stuba.sk](mailto:jozef.csambal@stuba.sk)

Marek Honek

Institute of Measurement, Automation and Informatics, Faculty of Mechanical Engineering, Slovak University of Technology in Bratislava, e-mail: [marek.honek@stuba.sk](mailto:marek.honek@stuba.sk)

Morten Hovd

Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim, Norway, e-mail: [morten.hovd@itk.ntnu.no](mailto:morten.hovd@itk.ntnu.no)

Mikuláš Huba

Institute of Control and Industrial Informatics, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, e-mail: [mikulas.huba@stuba.sk](mailto:mikulas.huba@stuba.sk)

Johannes Jäschke

Department of Chemical Engineering, Norwegian University of Science and Technology, Trondheim, Norway, e-mail: [jaschke@chemeng.ntnu.no](mailto:jaschke@chemeng.ntnu.no)

Matúš Kopačka

Institute of Measurement, Automation and Informatics, Faculty of Mechanical Engineering, Slovak University of Technology in Bratislava, e-mail: [matus.kopacka@stuba.sk](mailto:matus.kopacka@stuba.sk)

Michal Kvasnica

Institute of Information Engineering, Automation and Mathematics,  
Faculty of Chemical and Food Technology, Slovak University of Technology  
in Bratislava, e-mail: [michal.kvasnica@stuba.sk](mailto:michal.kvasnica@stuba.sk)

Tomáš Polóni

Institute of Measurement, Automation and Informatics, Faculty of Me-  
chanical Engineering, Slovak University of Technology in Bratislava, e-mail:  
[tomas.poloni@stuba.sk](mailto:tomas.poloni@stuba.sk)

Boris Rohal'-Ilkiv

Institute of Measurement, Automation and Informatics, Faculty of Me-  
chanical Engineering, Slovak University of Technology in Bratislava, e-mail:  
[boris.rohal-ilkiv@stuba.sk](mailto:boris.rohal-ilkiv@stuba.sk)

Peter Šimončíč

Institute of Measurement, Automation and Informatics, Faculty of Me-  
chanical Engineering, Slovak University of Technology in Bratislava, e-mail:  
[peter.simoncic@stuba.sk](mailto:peter.simoncic@stuba.sk)

Selvanathan Sivalingam

Department of Engineering Cybernetics, Norwegian University of Science and  
Technology, Trondheim, Norway, e-mail: [selvanathan.sivalingam@itk.ntnu.no](mailto:selvanathan.sivalingam@itk.ntnu.no)

Sigurd Skogestad

Department of Chemical Engineering, Norwegian University of Science and  
Technology, Trondheim, Norway, e-mail: [skoge@chemeng.ntnu.no](mailto:skoge@chemeng.ntnu.no)

Gergely Takács

Institute of Measurement, Automation and Informatics, Faculty of Me-  
chanical Engineering, Slovak University of Technology in Bratislava, e-mail:  
[gergely.takacs@stuba.sk](mailto:gergely.takacs@stuba.sk)

Peter Ťapák

Institute of Control and Industrial Informatics, Faculty of Electrical  
Engineering and Information Technology, Slovak University of Technology  
in Bratislava, e-mail: [peter.tapak@stuba.sk](mailto:peter.tapak@stuba.sk)

Slavomír Wojnar

Institute of Measurement, Automation and Informatics, Faculty of Me-  
chanical Engineering, Slovak University of Technology in Bratislava, e-mail:  
[slawomir.wojnar@stuba.sk](mailto:slawomir.wojnar@stuba.sk)

Ramprasad Yelchuru

Department of Chemical Engineering, Norwegian Univer-  
sity of Science and Technology, Trondheim, Norway, e-mail:  
[ramprasad.yelchuru@chemeng.ntnu.no](mailto:ramprasad.yelchuru@chemeng.ntnu.no)

Vladimír Žilka

Institute of Control and Industrial Informatics, Faculty of Electrical  
Engineering and Information Technology, Slovak University of Technology  
in Bratislava, e-mail: [vladimir.zilka@stuba.sk](mailto:vladimir.zilka@stuba.sk)

# Chapter 1

## Problems in Anti-Windup and Controller Performance Monitoring

Morten Hovd and Selvanathan Sivalingam

**Abstract** This chapter provides assignments for control engineering students on the topics of anti-windup and controller performance monitoring. The assignments are provided with detailed problem setups and solution manuals. Windup has been recognized for decades as a serious problem in control applications, and knowledge of remedies for this problem (i.e., anti-windup techniques) is essential knowledge for control engineers. The first problem in this chapter will allow students to acquire and apply such knowledge. The subsequent problems focus on different aspects of Controller Performance Monitoring (CPM), an area that has seen rapid developments over the last two decades. CPM techniques are important in particular for engineers working with large-scale plants with a large number of control loops. Such plants are often found in the chemical processing industries.

### 1.1 Anti-Windup: Control of a Distillation Column with Input Constraints

This assignment lets the student apply three different controller design methods to the control of a  $2 \times 2$  distillation column model. Subsequently, the controller implementations should be modified to account for input constraints.

---

Morten Hovd  
Department of Engineering Cybernetics, Norwegian University of Science and Technology, e-mail: [morten.hovd@itk.ntnu.no](mailto:morten.hovd@itk.ntnu.no)

Selvanathan Sivalingam  
Department of Engineering Cybernetics, Norwegian University of Science and Technology, e-mail: [morten.hovd@itk.ntnu.no](mailto:morten.hovd@itk.ntnu.no)



### 1.1.1 Notation

Consider a linear continuous-time state space model given by

$$\begin{aligned}\dot{x} &= Ax + Bu \\ y &= Cx + Du\end{aligned}$$

where  $x$  is the state vector,  $u$  is the input vector, and  $y$  is the output vector, and  $A, B, C, D$  are matrices of appropriate dimension. The corresponding transfer function model is given by

$$G(s) = C(sI - A)^{-1}B + D$$

The following equivalent shorthand notation is adopted from [Skogestad and Postlethwaite \(2005\)](#), and will be used when convenient

$$G(s) = \left[ \begin{array}{c|c} A & B \\ \hline C & D \end{array} \right]$$

### 1.1.2 Some Background Material on Anti-windup

In virtually all practical control problems, the range of actuation for the control input is limited. Whenever the input reaches the end of its range of actuation (the control input is *saturated*), the feedback path is broken. If the controller has been designed and implemented without regard for this problem, the controller will continue operating as if the inputs have unlimited range of actuation, but further increases in the controller output will not be implemented on the plant. The result may be that there is a large discrepancy between the internal states of the controller and the input actually applied to the plant. This problem often persists even after the controlled variable has been brought back near its reference value, and controllers that would work fine with unlimited inputs or with small disturbances, may show very poor performance once saturation is encountered.

The problem described is typically most severe when the controller has slow dynamics – integral action is particularly at risk (since a pure integration corresponds to a time constant of infinity). An alternative term for integral action is *reset action*, since the integral action ‘resets’ the controlled variable to its reference value at steady state. When the input saturates while there remains an offset in the controlled variable, the integral term will just continue growing, it ‘winds up’. The problem described above is therefore often termed *reset windup*, and remedial action is correspondingly termed *anti-reset windup* or simply *anti-windup*.

Anti-windup techniques remain an active research area, and no attempt is made here to give a comprehensive review of this research field. The aim is rather to present some important and useful techniques that should be known to practising control engineers.

### 1.1.2.1 Simple PI Control Anti-windup

A simple PI controller with limited actuation range for the control inputs (i.e., controller *outputs*), may be implemented as illustrated in Fig. 1.1. Here, the actual input implemented on the plant is feed back to the controller through the low pass filter  $1/(\tau_I s + 1)$ . If the actual plant input is not measured, it suffices to know the range of actuation for the input. The actual input can then easily be calculated.

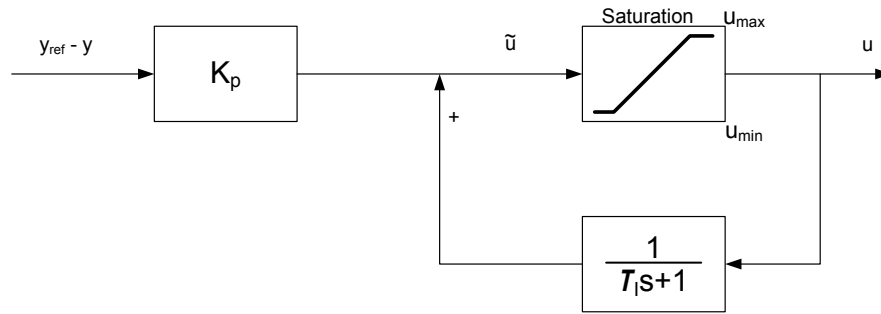


Fig. 1.1 Simple anti-windup scheme for a PI controller.

From Fig. 1.1, it is easy to see that when the plant input is not saturated (when  $\tilde{u} = u$ ), we get

$$u = K_p \frac{\tau_I s + 1}{\tau_I s} (y_{ref} - y) \quad (1.1)$$

That is, we get the normal behaviour of a PI controller. On the other hand, consider the case when the input is in saturation at its upper limit  $u_{max}$ :

$$\tilde{u} = K(y_{ref} - y) + \frac{1}{\tau_I s + 1} u_{max} \quad (1.2)$$

The internal feedback path in the controller is now broken, there is no open integrator in the controller, and the controller state goes to  $u_{max}$  with a time constant  $\tau_I$ . Thus, the integrating state does not wind up. Note also that when the controller state has reached its stationary value of  $u_{max}$ , the

controller output will stay at its maximum value until the measurement  $y$  has crossed the reference value  $y_{ref}$ .

This anti-windup scheme is straight forward and simple to implement provided any actuator dynamics is fast compared to the PI controller time constant  $\tau_I$ .

### 1.1.2.2 Velocity Form of PI Controllers

The PI controller in (1.1) is in *position form*, i.e., the controller output corresponds to the desired position/value of the plant input. Alternatively, the controller output may give the desired *change* in the plant input.

Whereas the equations for PI controllers in position form are often expressed in continuous time (even though the final implementation in a plant computer will be in discrete time), the velocity form of the PI controller is most often expressed in discrete time. Let the subscript denote the discrete time index, and  $e_k = y_{ref} - y_k$  be the control offset at time  $k$ . The discrete time equivalent of (1.1) may then be expressed as

$$\Delta u_k = u_k - u_{k-1} = \frac{T}{\tau_I} e_{k-1} + K_p(e_k - e_{k-1}) \quad (1.3)$$

where  $T$  is the sample interval. Here  $\Delta u_k$  represents the *change* in the plant input at time  $k$ . If this change is sent to the actuator for the plant input, instead of the desired position of the input, the windup problem goes away. This is because desired changes that violate the actuation constraints simply will not have any effect. Note that the actuator should implement *new desired value = present value +  $\Delta u_k$*

If the *previous desired value* is used instead of *present value* above, the velocity form of the controller will not remove windup problems.

The velocity form can also be found for more complex controllers, in particular for PID controllers. However, derivative action is normally rather fast, and the effects thereof quickly die out. It is therefore often not considered necessary to account for the derivative action in anti-windup of PID controllers.

### 1.1.2.3 Anti-windup in Cascaded Control Systems

For ordinary plant input, it is usually simple to determine the range of actuation. For instance, a valve opening is constrained to be within 0 and 100%, maximum and minimum operating speeds for pumps are often well known, etc. In the case of cascaded control loops, the 'plant input' seen by the outer loop is actually the reference signal to the inner loop, and the control is typically based on the assumption that the inner loop is able to follow the reference changes set by the outer loop. In such cases, the 'available range of

actuation' for the outer loop may be harder to determine, and may depend on operating conditions. An example of this problem may be a temperature control system, where the temperature control loop is the outer loop, and the inner loop is a cooling water flow control loop with the valve opening as the plant input. In such an example, the maximum achievable flowrate may depend on up- and downstream pressures, which may depend on cooling water demand elsewhere in the system.

Possible ways of handling anti-windup of the outer loop in such a situation include

- Using conservative estimates of the available range of actuation, with the possibility of not fully utilising plant capacity in some operating scenarios.
- The controller in the inner loop may send a signal informing the controller in the outer loop when it is in saturation (and whether it is at its maximum or minimum value). The controller in the outer loop may then stop the integration if this would move the controller output in the wrong direction.
- Use the velocity form of the controller, provided the reference signal for the inner loop is calculated as *present plant output + change in reference from outer loop*. If the reference signal is calculated as 'reference at last time step + change in reference from outer loop', windup may still occur.
- For PI controllers, use the implementation shown in Fig. 1.1, where the 'plant input' used in the outer loop is the plant measurement for the inner loop.

Note that the two latter anti-windup schemes above both require a clear timescale separation between the loops (the inner loop being faster than the outer loop), otherwise performance may suffer when the plant input (in the inner loop) is not in saturation. There is usually a clear timescale separation between cascaded loops.

#### 1.1.2.4 Hanus' Self-conditioned Form

Hanus' self-conditioned form (Hanus et al (1987); Skogestad and Postlethwaite (2005)) is a quite general way of preventing windup in controllers. Assume a linear controller is used, with state space realization

$$\dot{v} = A_K v + B_K e \quad (1.4)$$

$$\tilde{u} = C_K v + D_K e \quad (1.5)$$

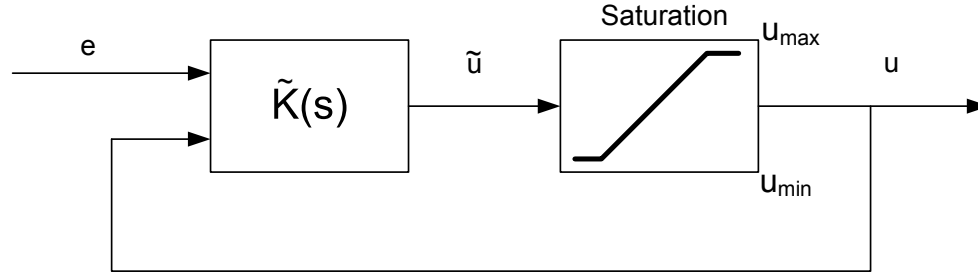
where  $v$  are the controller states,  $e$  are the (ordinary) controller inputs, and  $\tilde{u}$  is the calculated output from the controller (desired plant input). The corresponding controller transfer function may be expressed as

$$K(s) \stackrel{s}{=} \left[ \begin{array}{c|c} A_K & B_K \\ \hline C_K & D_K \end{array} \right] = C_K (sI - A_K)^{-1} B_K + D_K \quad (1.6)$$

The corresponding implementation of the same controller in Hanus' self-conditioned form is illustrated in Fig. 1.2, where  $\tilde{K}(s)$  given by

$$\tilde{u} = \tilde{K}(s) \begin{bmatrix} e \\ u \end{bmatrix}$$

$$K(s) \stackrel{s}{=} \left[ \begin{array}{c|cc} A_K - B_K D_K^{-1} C_K & 0 & B_K D_K^{-1} \\ \hline C_K & D_K & 0 \end{array} \right] \quad (1.7)$$



**Fig. 1.2** Illustration of anti-windup with the controller  $K(s)$  implemented in its self-conditioned form  $\tilde{K}(s)$ .

From (1.7) we see that when the plant input  $u$  is not saturated, i.e., when  $\tilde{u} = u$ , the controller dynamics are given by (1.5). When the plant input is saturated, the steady state controller output will be

$$\tilde{u} = -C_K(A_K - B_K D_K^{-1} C_K)^{-1} u + D_K e \quad (1.8)$$

If  $B_K D_K^{-1} C_K \gg A_K$ , we get

$$\tilde{u} \approx u + D_K e \quad (1.9)$$

and thus the plant input will stay at its limit until the corresponding element of  $D_K e$  changes sign.

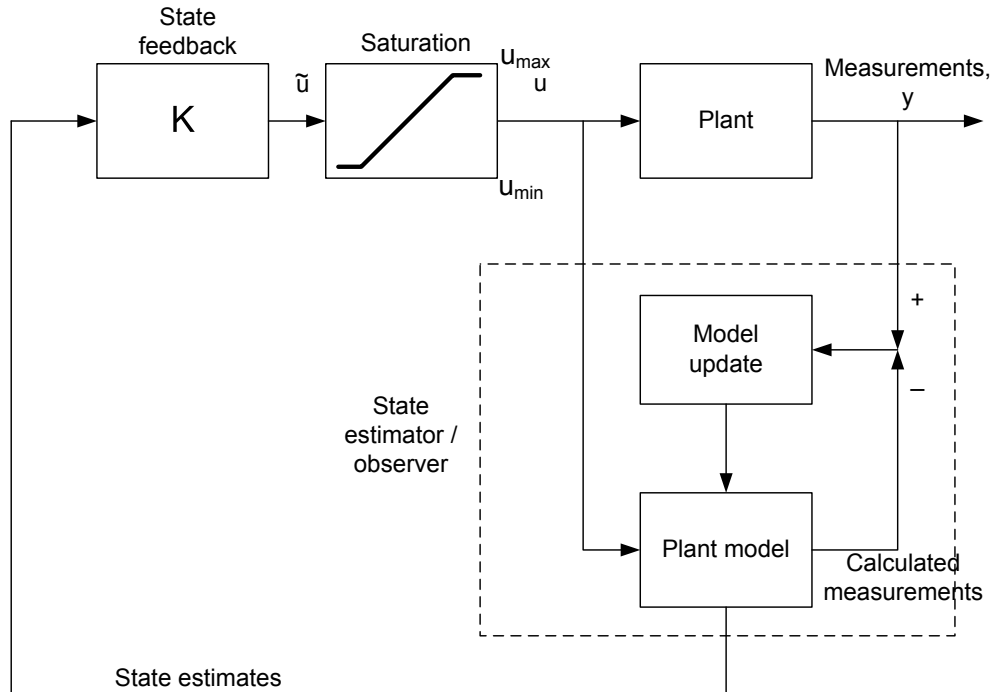
Clearly, the use of this anti-windup methodology requires  $D_K$  to be invertible, and hence also of full rank. Thus, the controller must be semi-proper. The rate at which the controller states converge towards the steady state solution (when in saturation) is given by the eigenvalues of  $A_K - B_K D_K^{-1} C_K$ . This matrix obviously has to be stable. A small (but non-singular)  $D_K$  will generally make the convergence fast.

In Hanus et al (1987), self-conditioning is presented in a more general setting, potentially accounting also for time-varying or non-linear controllers. However, only in the case of linear time-invariant controllers do the resulting controller equations come out in a relatively simple form.

### 1.1.2.5 Anti-windup in Observer-based Controllers

Many advanced controllers are (or may be) implemented as a combination of static state feedback controllers and a state observer/estimator. This is the case for  $LQG/H_2$ -optimal controllers as well as  $H_\infty$ -optimal controllers.

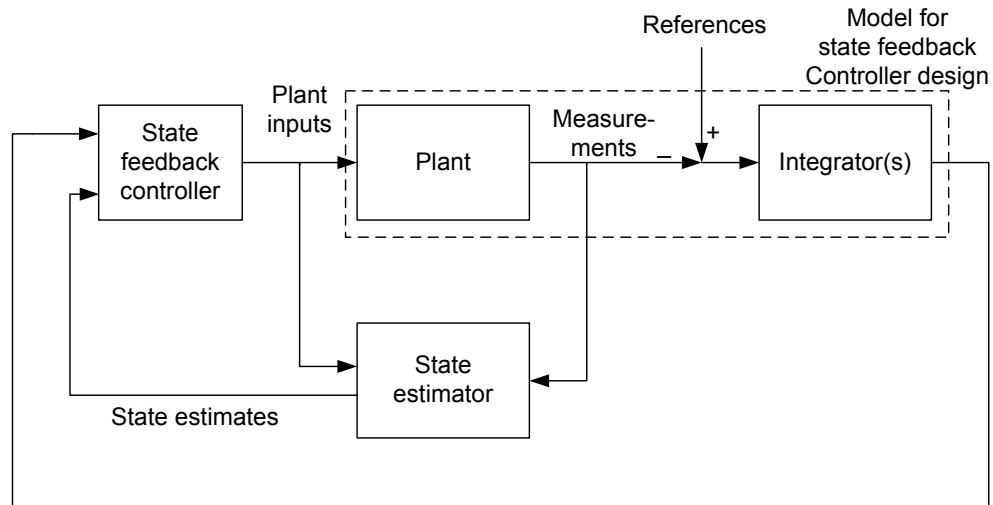
For such controllers, anti-windup is achieved by ensuring that the state observer/estimator receives the *actual plant input that is implemented on the plant*. This is illustrated in Fig. 1.3



**Fig. 1.3** Illustration of anti-windup for controllers based on static state feedback combined with state estimation.

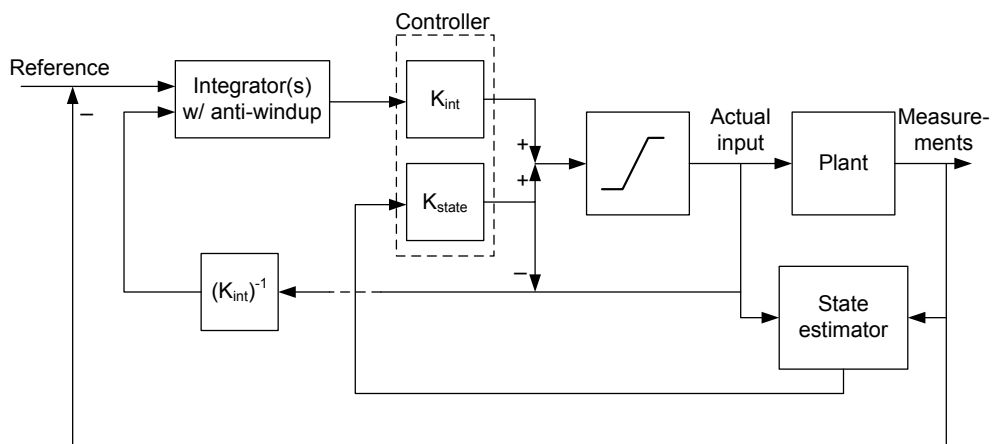
In many applications it is desired to have offset-free control at steady state. This requires the use of integral action. This is often incorporated in a state estimator/state feedback control design as illustrated in Fig. 1.4.

The state estimator only estimates actual plant states, whereas the state feedback is designed for a model where integrators (which integrate the control offset) are appended to the plant model. When implementing the controller, the integrators are a part of the controller (in the control system). The values of the integrators are thus directly available in the control system, and clearly there is no need to estimate these states.



**Fig. 1.4** State estimator and static state feedback augmented with integral action.

However, when integration is incorporated in this way, the integrating states may wind up even if the actual input values are sent to the state estimator. Fig. 1.5 illustrates how the anti-windup signal to the integrators must represent the range of movement available for the integrating states, i.e., with the contribution from the (actual) state feedback removed.



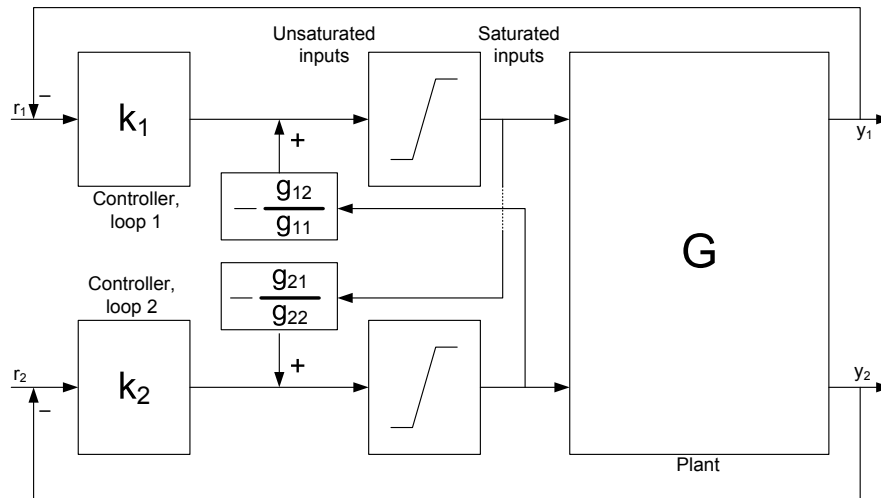
**Fig. 1.5** Implementation of anti-windup for state estimator and static state feedback augmented with integral action.

**Remark.** Note that if Hanus' self-conditioned form is used for the anti-windup, this requires a non-singular  $D$ -matrix, resulting in a PI block instead of a purely integrating block. The size of this  $D$ -matrix may affect controller performance (depending on how and whether it is accounted for in the 'state' feedback control design).

### 1.1.3 Decoupling and Input Constraints

Decouplers are particularly prone to performance problems due to input constraints. This is not easily handled by standard anti-windup, because much of the input usage can be related to counteracting interactions. Therefore, if an output is saturated, but other outputs are adjusted to counteract the effects of the 'unsaturated' output, severe performance problems may be expected.

One way of ensuring that the decoupler only tries to counteract interactions due to the inputs that are actually implemented on the plant, is to implement the decoupler as illustrated in Fig. 1.6.



**Fig. 1.6** Implementation of decoupler in order to reduce the effect of input saturation. The decoupler will only attempt to counteract interactions due to inputs that are actually implemented on the plant.

The implementation in Fig. 1.6 is easily extended to systems of dimension higher than  $2 \times 2$ . When the inputs are unsaturated, the 'Decoupler with saturation' in Fig. 1.1 corresponds to the decoupling compensator  $W(s) = G(s)^{-1}\tilde{G}(s)$ , where  $\tilde{G}(s)$  denotes the diagonal matrix with the same diagonal



elements as  $G(s)$ . The precompensated plant therefore becomes  $GW = \tilde{G}$ , i.e., we are (nominally) left only with the diagonal elements of the plant.

Note that if the individual loop controllers  $k_i(s)$  contain slow dynamics (which is usually the case, PI controllers are often used), they will still need anti-windup. In this case the anti-windup signal to the controller should not be the saturated input, but the saturated input *with the contribution from the decoupling removed*, i.e., the decoupling means that the saturation limitations for the individual loop controllers  $k_i(s)$  are time variant.

#### 1.1.4 The Plant Model used in the Assignment

The model used is the detailed LV model of a distillation column given in (13.19) of [Skogestad and Postlethwaite \(2005\)](#):

$$y(s) = \begin{bmatrix} A & B & B_d \\ C & D & D_d \end{bmatrix} \begin{bmatrix} u(s) \\ d(s) \end{bmatrix} \quad (1.10)$$

$$A = \begin{bmatrix} -0.005131 & 0 & 0 & 0 & 0 \\ 0 & -0.07366 & 0 & 0 & 0 \\ 0 & 0 & -0.1829 & 0 & 0 \\ 0 & 0 & 0 & -0.4620 & 0.9895 \\ 0 & 0 & 0 & -0.9895 & -0.4620 \end{bmatrix}$$

$$B = \begin{bmatrix} -0.629 & 0.624 \\ 0.055 & -0.172 \\ 0.030 & -0.108 \\ -0.186 & -0.139 \\ -1.230 & -0.056 \end{bmatrix}$$

$$C = \begin{bmatrix} -0.7223 & -0.5170 & 0.3386 & -0.1633 & 0.1121 \\ -0.8913 & -0.4728 & 0.9876 & 0.8425 & 0.2186 \end{bmatrix}, \quad D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$B_d = \begin{bmatrix} -0.062 & -0.067 \\ 0.131 & 0.040 \\ 0.022 & -0.106 \\ -0.188 & 0.027 \\ -0.045 & 0.014 \end{bmatrix}, \quad D_d = D$$

where the two measurements are the top and bottom product concentrations, the first input is the reflux flowrate at the top of the distillation column, the second input is the boilup rate at the bottom of the distillation column, the first disturbance is the feed flowrate, and the second disturbance is the feed composition. The time scale in the model is in minutes. For further information about the model and the scalings applied, please refer to [Skogestad and Postlethwaite \(2005\)](#).

### 1.1.5 Assignment

#### 1. Model implementation

Implement the LV model in Simulink.

Use `saturation` blocks (found in the `discontinuities` library) of  $\pm 0.5$  for the inputs to the LV model. You may use `manual switch` blocks (found in the `signal routing` library) to make it possible to choose whether the saturation should be turned on or off for the simulations. The disturbance input is modelled as white noise (independently normally distributed random variables) with variance 0.001. The measurement noise is white noise with variance 0.0004.

#### 2. Controller design

Design three different controllers for the model:

- a. A controller based on two PI/PID loops.
- b. A controller based on dynamic decoupling.
- c. A controller based on multi-variable controller synthesis (LQG/ $H_2$ / $H_\infty$ ).

All the controllers should be designed to give zero stationary deviation. The dominant time constant for the closed loop should be around one minute.

PI/PID controller design should be well known. The multi-variable controller design method chosen is also assumed known – all the three controller synthesis methods proposed above are of the state estimator / state feedback type. Decoupling is also a relatively common controller design, see [Wade \(1997\)](#) for a thorough description of decoupling for  $2 \times 2$  systems if necessary.

#### 3. Simulation (with disturbances)

Simulate each of the three controllers in Simulink.

Use the following reference signals:

$y_{1,ref}$ : square pulse, amplitude 1, frequency 0.005Hz

$y_{2,ref}$ : 0

Simulate each controller design both without and with saturation in the inputs. Discuss the results of the simulation.

#### 4. Anti-windup (and simulation)

Implement anti-windup for all three controllers.

Simulate with the reference changes and noise variances prescribed above. Plot the results and compare to the performance without anti-windup (make comparable plots with and without anti-windup implemented). Comment on the results.

### 1.1.6 Solution

#### 1.1.6.1 Model Implementation

The model is implemented in Simulink as illustrated in Fig. 1.7. Note that the state-space model for the distillation column has been implemented with the disturbance as the second input, in accordance with (1.10). A manual switch is used to include/exclude saturation of the manipulated variable, and clearly this switch will have to be set in accordance with the case to be simulated.

The Matlab code to define the model is listed in Section 1.1.9.1.

#### 1.1.6.2 Controller Designs

##### PI Controllers

The PI controllers without anti-windup are implemented using transfer function blocks in Simulink, cf. Fig. 1.8. Alternatively, the PID Controller block in the 'Continuous' library can be used, but one should be aware that the tuning parameters in that block does not correspond to the 'standard' proportional gain and integral time parameters (consult Simulink documentation for details).

The Matlab code used to define the controller numerator and denominator polynomials are given in subsection 1.1.9.2. Note that the sign of the proportional gain is adjusted to ensure negative feedback.

##### Dynamic Decoupler

The dynamic decoupler is here implemented in two different ways, both as a 'standard' decoupler  $W(s) = G^{-1}(s)\tilde{G}(s)$ , where  $\tilde{G}(s)$  is the diagonal matrix corresponding to the diagonal elements of  $G(s)$ , and using the inverted decoupler structure shown in Fig. 1.6. Note that  $G(s)$  here represents the transfer function matrix from manipulated variables (only) to the measurements, i.e.,

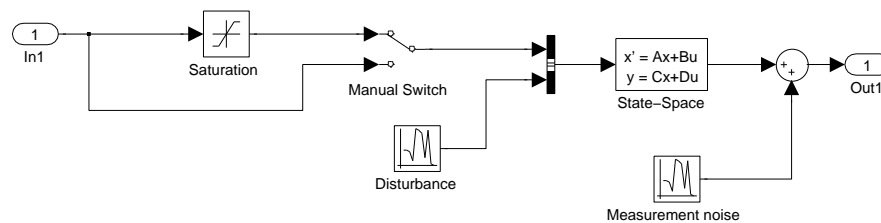


Fig. 1.7 Implementation of distillation column model in Simulink

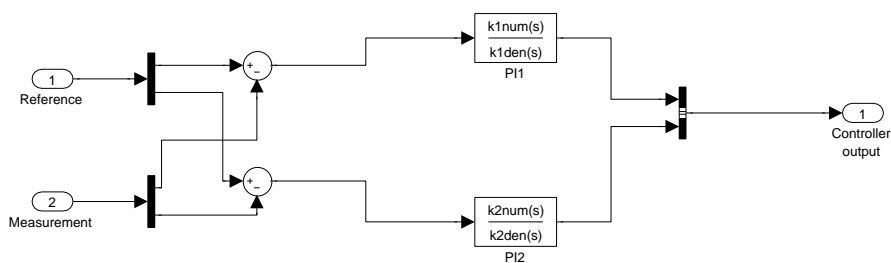
without the disturbance inputs. Note also that in order to invert  $G(s)$ , a full rank  $D$  matrix is necessary. Small diagonal elements are therefore added to the  $D$  matrix – and it will be necessary to check that these do not introduce right half plane zeros, which would lead to unstable poles after system inversion.

The Matlab Control Systems Toolbox allows using the command `inv(G)` to find the inverse of  $G(s)$ . Alternatively, a state space formula for the inverse can be found in [Skogestad and Postlethwaite \(2005\)](#).

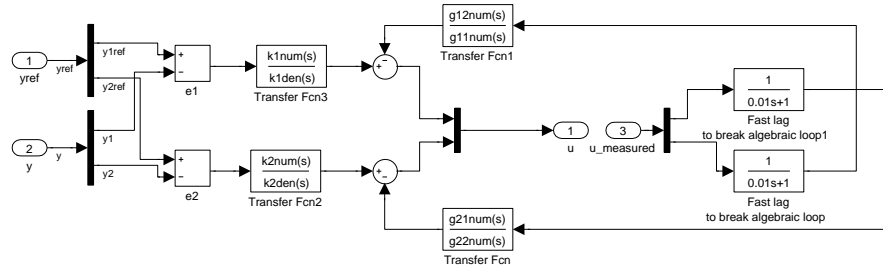
The 'standard' decoupler is simulated by simply putting the PI controllers and the decoupler  $W(s)$  (represented by a state space model block) in series. The inverted decoupler with PI controllers is shown in Fig. 1.9. Note that anti-windup of the PI controllers is not included in this figure. Note also that fast low-pass filters are introduced in order to break so-called 'algebraic loops' in the Simulink simulation. Such algebraic loops are a problem only in simulation, not in a physical implementation. If the low-pass filters are significantly faster than the rest of the dynamics in the system, they should have little effect on simulation results. However, *very* fast low-pass filters could lead to a stiff simulation problem, causing long simulation times or requiring the use of special purpose integration routines.

### LQG Controller

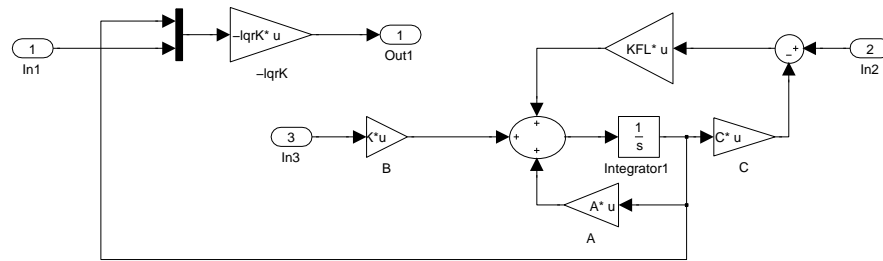
The Simulink setup for the LQG controller is shown in Fig. 1.10. The Kalman filter part of the LQG controller is designed with the physical model states, and the specified noise covariances. The LQ regulator part of the LQG controller is designed with a model that is augmented with integrators in order to include integral action. The weights used in the LQ design are found by trial and error in order to (approximately) achieve the specified closed loop bandwidth. The overall simulation setup, including integrating controller states, LQG controller and distillation column model, is shown in Fig. 1.11 Note that provided the Kalman filter is provided with the saturated input values, it will provide good state estimates also when the inputs are saturated. However,



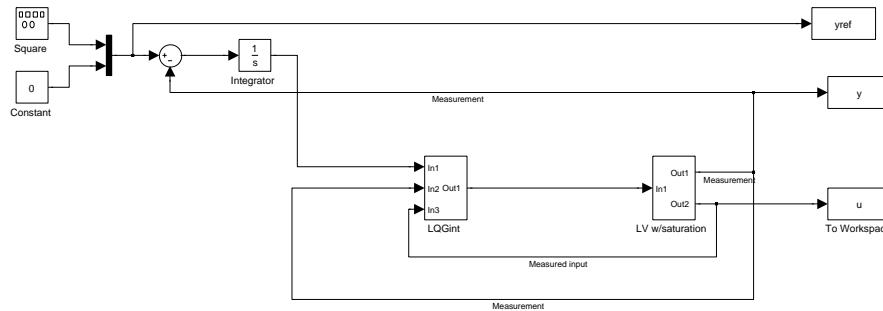
**Fig. 1.8** Implementation of PI controllers without anti-windup.



**Fig. 1.9** Simulink implementation of inverted decoupler, without anti-windup of PI controllers



**Fig. 1.10** Simulink implementation of LQG controller. Input 1 are the values of the augmented integrating states (which are available to the controller), whereas input 2 is the measurement vector.



**Fig. 1.11** Overall simulation setup for LQG controller with integrating states.

no anti-windup is provided for the integrating states of the (augmented) LQ regulator.

### 1.1.7 Simulations

#### 1.1.7.1 PI Controller

The responses in the controlled outputs, when no saturation is present are shown in Fig. 1.12. We observe that the setpoint changes are followed quickly. However, there is some interaction between the two loops, causing offset in loop 2 following setpoint changes in loop 1. These interactions are counteracted relatively quickly by the controllers.

The corresponding responses when the input is constrained between  $\pm 0.5$  are shown in Fig. 1.13. It is obvious that the saturation severely degrades control performance.

#### 1.1.7.2 Decoupler

The responses in the controlled outputs, when no saturation is present, are shown in Fig. 1.14. The observed performance is good (and could probably have been further improved by more careful design of the PI controllers), and the decoupling is practically perfect, with no observable interaction between the two outputs. Note that these results do not depend on whether a standard or an inverted decoupler implementation is used.

The response with the standard decoupler when the input is constrained between  $\pm 0.5$  are shown in Fig. 1.15. Clearly, performance is horrible, and the interactions between the outputs extreme. Note, however, that the system (in

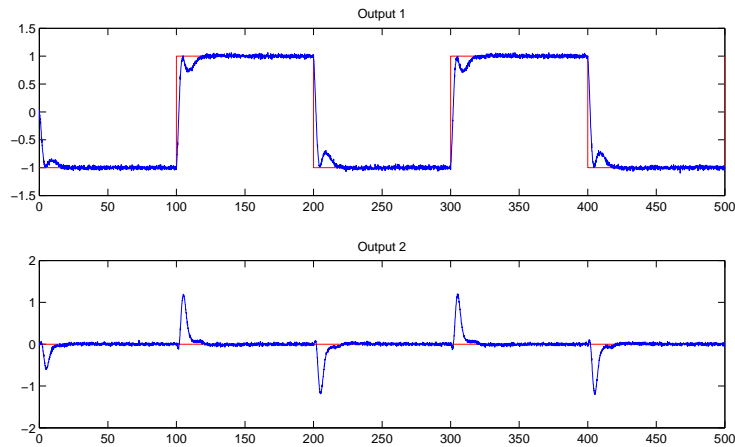
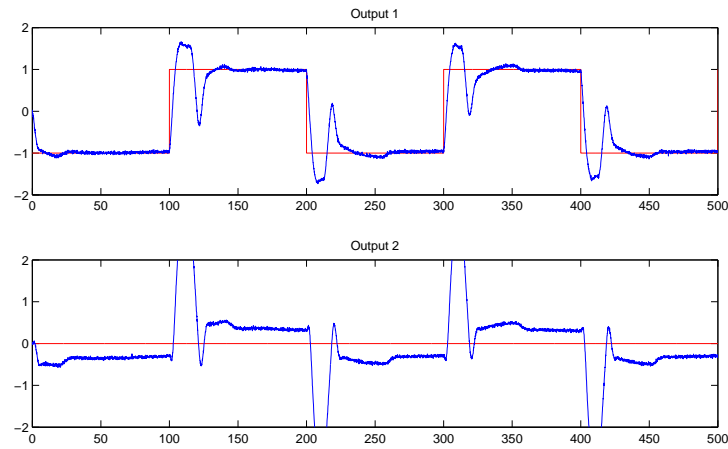
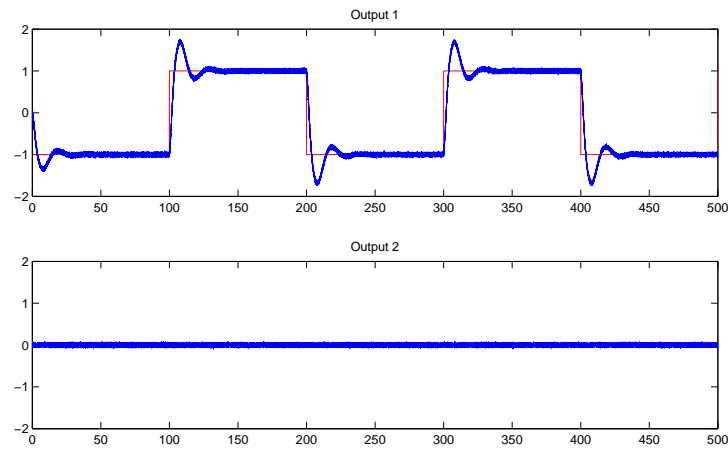


Fig. 1.12 Responses with PI controllers, in the absence of saturation.



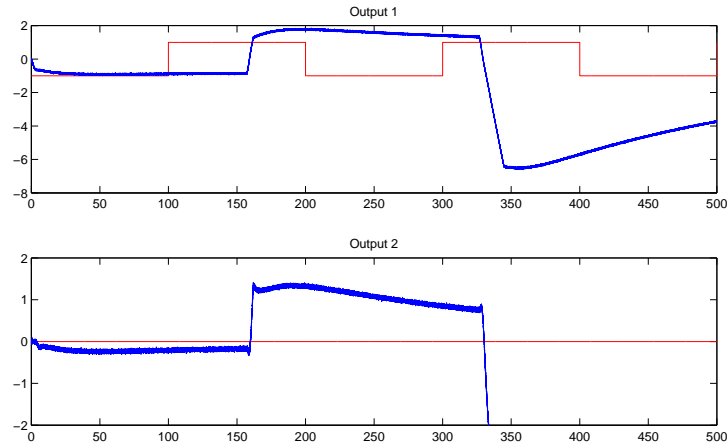
**Fig. 1.13** Responses with PI controllers, with saturation in the inputs.



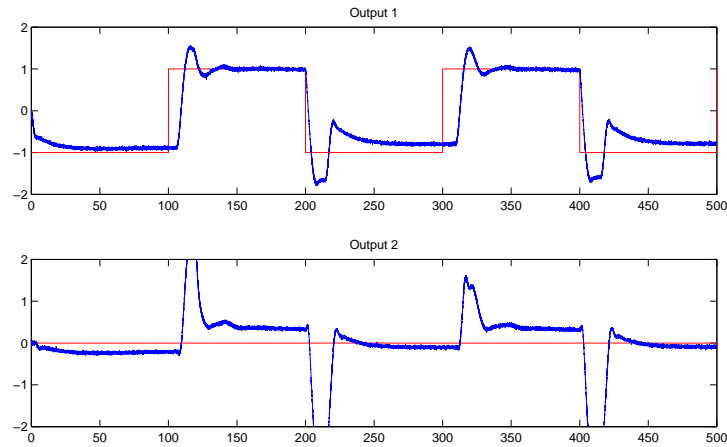
**Fig. 1.14** Responses with decoupler, in the absence of saturation.

this case) remains stable – although the responses leave the window displayed in the figure.

The corresponding response with the inverted decoupler are shown in Fig. 1.16. We see that the performance is much better than with the standard decoupler, although there is still significant interactions between the outputs. These responses still suffer from the absence of anti-windup for the PI controllers used for the individual loops.



**Fig. 1.15** Responses with the standard decoupler, with saturation in the inputs.



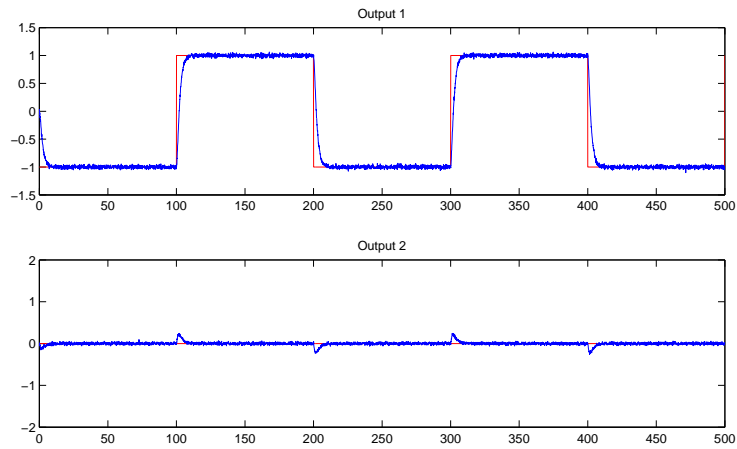
**Fig. 1.16** Responses with the inverted decoupler, with saturation in the inputs.

### 1.1.7.3 LQG Controller with Integral Action

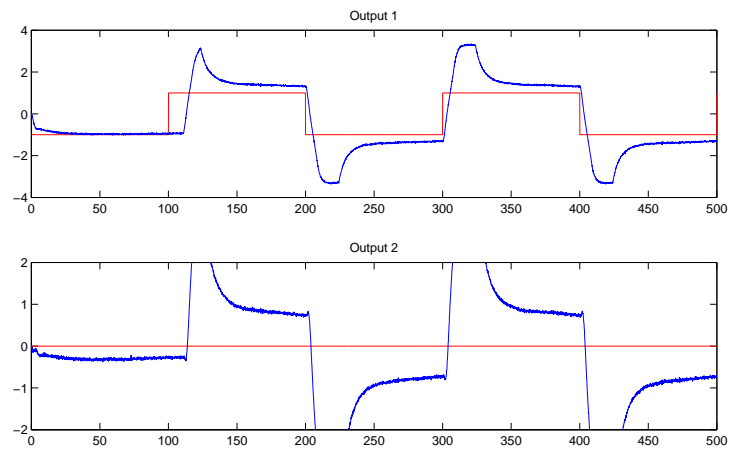
The responses in the outputs when using the LQG controller are shown in Fig. 1.17. We observe that the performance is good, and there is very little interaction between the outputs.

The corresponding responses when the input is constrained between  $\pm 0.5$  are shown in Fig. 1.18. Note that the Kalman filter is here supplied the



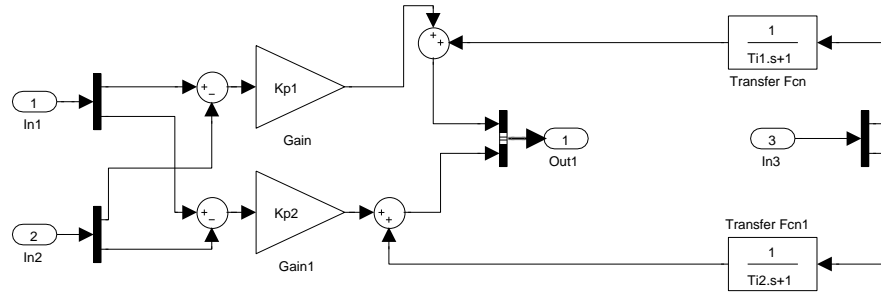


**Fig. 1.17** Responses with the LQG controller with integral action, in the absence of saturation.



**Fig. 1.18** Responses with the LQG controller with integral action, with saturation in the inputs.

actual (constrained, saturated) input values, but there is no anti-windup for the augmented integrators. The performance is significantly poorer than for the case without saturation.



**Fig. 1.19** Implementation of two PI controllers with simple anti-windup technique in Simulink.

#### 1.1.7.4 Conclusion on Simulation Results

The simulations show that all three controller types are able to achieve good control of the plant. For all three controllers, the performance is significantly degraded in the presence of saturation, although to different degrees for the different controllers.

### 1.1.8 Anti-windup

In this section, anti-windup is implemented for all three controllers, and the simulations re-run.

#### 1.1.8.1 PI Controllers

The simple PI anti-windup scheme in Fig. 1.1 is used. This simple anti-windup scheme is implemented in Simulink as illustrated in Fig. 1.19. The figure shows the implementation of both PI controllers, where input 1 is the reference, input 2 is the measurement, and input 3 is the actual (measured) values of the manipulated variables. The results when using this anti-windup technique are shown in Fig. 1.20. We see that the performance is markedly improved due to anti-windup, although output 2 is not able to return to its setpoint due to saturation.

#### 1.1.8.2 Decoupler

Although the inverse decoupler reduces the problems of input saturation compared to ordinary decoupling, anti-windup is still necessary if integral action

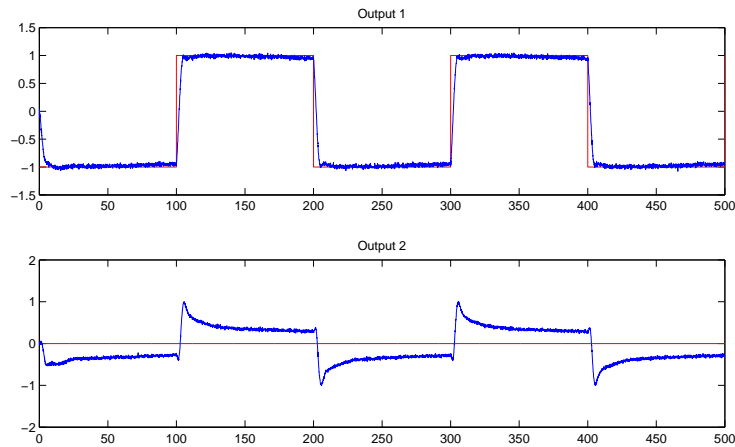


Fig. 1.20 Response with PI controllers with anti-windup.

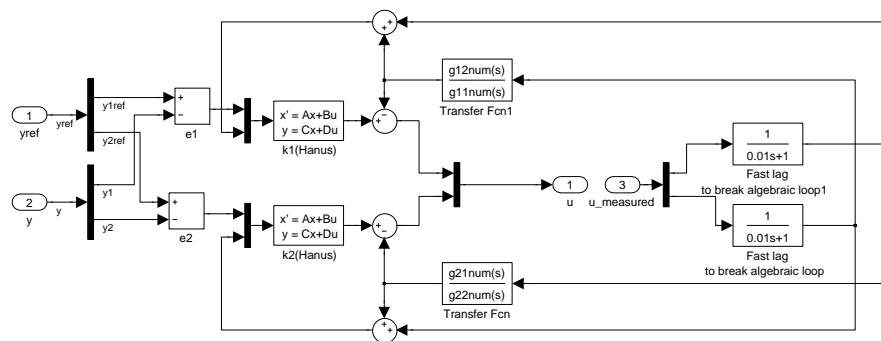
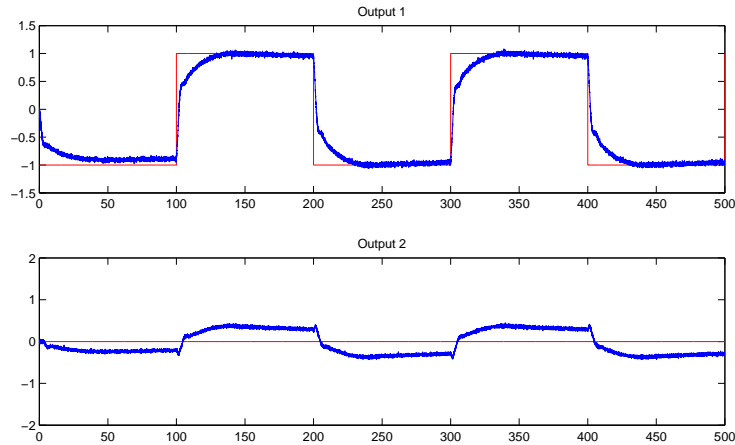


Fig. 1.21 Simulink implementation of inverse decoupler with anti-windup for outer PI controllers.

is used in the controllers for the individual loops that result after decoupling compensation. This is the common case, which also is used in this example. However, even if simple PI controllers are used in the individual loops, we may can no longer use the simple anti-windup scheme in Fig. 1.1. Rather, we have to calculate the range of manipulated (input) variable movement that is available, *after* accounting for the action of the decoupling elements. This is illustrated by the Simulink implementation in Fig. 1.21. Note that here the anti-windup of the PI controllers has been achieved using the so-called Hanus form. This could equivalently have been done using the scheme in Fig. 1.1.

The resulting responses when using the inverted decoupler with anti-windup are shown in Fig. 1.22. Comparing to Fig. 1.16, we see that the



**Fig. 1.22** Responses with inverted decoupler with anti-windup.

responses are significantly improved, although far from as good as in the unconstrained case in Fig. 1.14.

### 1.1.8.3 LQG Controller with Integral Action

The LQG controller with integral action and anti-windup is implemented as illustrated in Fig. 1.5. The Simulink implementation is shown in Fig. 1.23. Note the fast low-pass filter used to break the 'algebraic loop' in the anti-windup loop of the controller. Due to the order of execution in digital control systems, this is not required in practice, but significantly simplifies numerical integration in continuous-time simulations. The time constant of these low-pass filters should be significantly shorter than any other dynamics in the system – but very short time constants may lead to a numerically 'stiff' system.

When anti-windup is included in the LQG controller with integral action, the results in Fig. 1.24 are obtained. The results are drastically improved compared to the case without anti-windup, and are quite similar to what was obtained for the decoupler with anti-windup above.

### 1.1.8.4 Conclusions on Anti-windup

The results obtained clearly illustrate the benefits of anti-windup when the manipulated variables saturate, for all three controller types studied in this assignment.

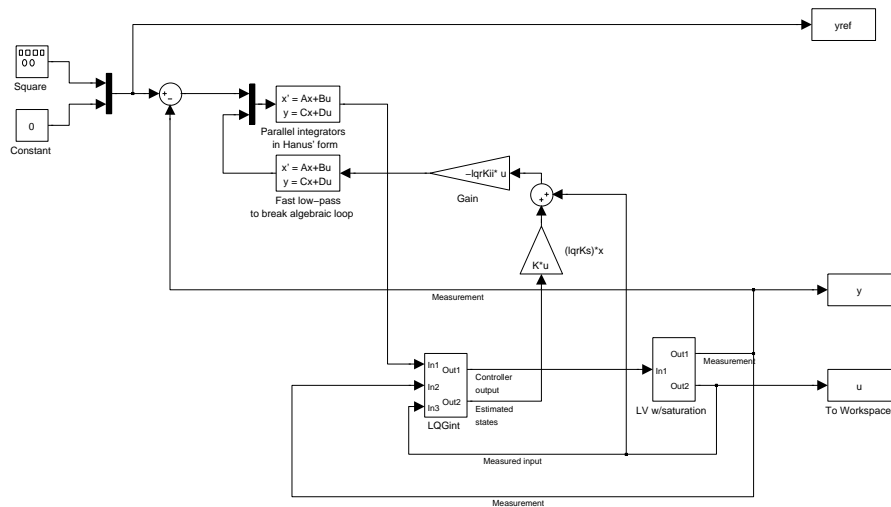
## 1.1.9 Matlab Code

### 1.1.9.1 Definition of Model and External Signals

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %                                                                 %
3  % Assignment: Controller design                                 %
4  %           and anti-windup - Solution                       %
5  %                                                                 %
6  % Model and signal definitions                               %
7  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
8  clc
9  clear all
10 close all
11

```



**Fig. 1.23** Simulink implementation of LV distillation column with LQG controller. The LQG controller has integral action (shown separately from the rest of the LQC controller) and anti-windup.

```

12 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
13 % Reference signals                                     %
14 % Parameters in signal generator blocks for %
15 % references                                           %
16 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
17 y1amp = 1;
18 y1freq = 0.005;
19 y2amp = 0;
20 y2freq = 0.005;
21
22 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
23 % System parameters                                     %
24 % Distillation column model                           %
25 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
26 A = [ -0.005131 0 0 0 0 ;
27        0 -0.07366 0 0 0 ;
28        0 0 -0.1829 0 0 ;
29        0 0 0 -0.4620 0.9895 ;
30        0 0 0 -0.9895 -0.4620 ];
31 B = [ -0.629 0.624 ;
32        0.055 -0.172 ;
33        0.030 -0.108 ;
34        -0.186 -0.139 ;
35        -1.230 -0.056 ];
36 C = [ -0.7223 -0.5170 0.3386 -0.1633 0.1121 ;

```

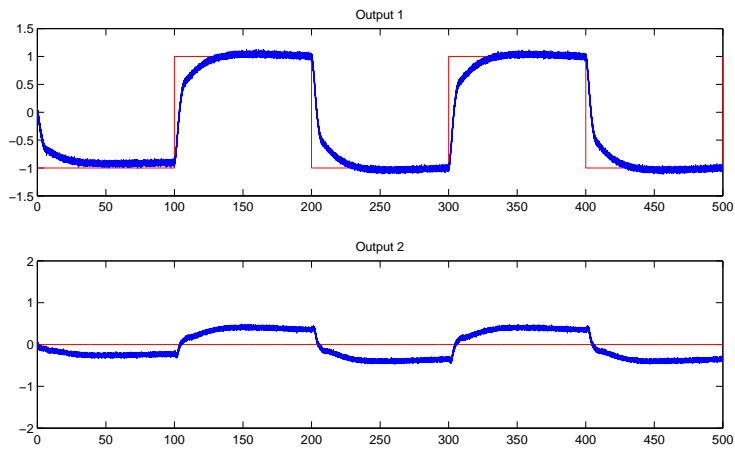


Fig. 1.24 Responses with LQG controller with integral action and anti-windup.

```

37         -0.8913 -0.4728 0.9876 0.8425 0.2186 ];
38 Bd    = [ -0.062 -0.067 ;
39           0.131 0.040 ;
40           0.022 -0.106 ;
41           -0.188 0.027 ;
42           -0.045 0.014 ];
43 D     = [ 0 0 ;
44           0 0 ];
45 sys   = ss(A,B,C,D);
46 G     = tf(sys);      % Used in decoupler
47
48 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
49 % White noise disturbances          %
50 % and measurement noise           %
51 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
52 kw    = 0.001;
53 Wvar  = kw*ones(2,1); %Disturbance variance
54 QXU   = eye(7);
55 kv    = 0.0004;
56 Vvar  = kv*ones(2,1); %Measurementn noise variance

```

### 1.1.9.2 Simple PI Controller without Anti-windup

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % PI controller          %
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 s = tf('s');
5 Kp1 = 0.5; Kp2 = -0.5; % Signs are decided through
6                       % dcgain(A,B,C,D), which results in
7                       % [88.3573 -86.8074; 108.0808 -107.9375]
8 Ti1 = 2; Ti2 = 2;
9 k1 = Kp1*(Ti1*s+1)/(Ti1*s); [k1num,k1den] = tfdata(k1); k1num
   = k1num{1,:}; k1den = k1den{1,:};
10 k2 = Kp2*(Ti2*s+1)/(Ti2*s); [k2num,k2den] = tfdata(k2); k2num
   = k2num{1,:}; k2den = k2den{1,:};

```

### 1.1.9.3 Decoupling

```

1 % Transfer function matrix G(s) defined in file model.m,
2 % which must be run before this file.

```

```

3 % Similarly, the individual PI controllers used with the
  decouplers are
4 % (in this case) identical to the controllers used for (
  ordinary) PI control. The file
5 % PIcont.m should therefore also run before this file
6
7 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
8 % Dynamic decoupling controller          %
9 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10 Gdiag = [G(1,1) 0      ;
11          0      G(2,2) ];
12 % We need a non-singular D-matrix to calculate inv(G).
13 % Find an approximate G using a new D-matrix that does not
14 % introduce poles in the right-half plane
15 dd  = [ 1e-3  0      ;
16         0     -1e-3 ];
17 Gapp = ss(A,B,C,dd);
18 %tzero(Gapp) % Check for zeros..
19 W     = inv(Gapp)*Gdiag;
20 W     = minreal(W);
21 [Wa,Wb,Wc,Wd] = ssdata(W);
22
23 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
24 % Anti-windup for dynamic decoupling      %
25 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
26 % Note that all elements share the same denominator
27 [g11num,g11den] = tfdata(sys(1,1));
28 g11den = g11den{1,:}; g11num = g11num{1,:};
29 [g12num,g12den] = tfdata(sys(1,2));
30 g12den = g12den{1,:}; g12num = g12num{1,:};
31 [g21num,g21den] = tfdata(sys(2,1));
32 g21den = g21den{1,:}; g21num = g21num{1,:};
33 [g22num,g22den] = tfdata(sys(2,2));
34 g22den = g22den{1,:}; g22num = g22num{1,:};
35
36 % Using Hanus' self-conditioned form for anti-windup
37 % of PI controllers
38 [k2a,k2b,k2c,k2d]=ssdata(k2);
39 [k1a,k1b,k1c,k1d]=ssdata(k1);
40
41 k1aw = k1a-k1b*inv(k1d)*k1c;
42 k1bw = [zeros(size(k1b)) k1b*inv(k1d)];
43 uz = size(k1b*inv(k1d));
44 k1cw = k1c;
45 k1dw = [k1d zeros(uz)];

```



```

46
47 k2aw = k2a-k2b*inv(k2d)*k2c;
48 k2bw = [zeros(size(k2b)) k2b*inv(k2d)];
49 uz = size(k2b*inv(k2d));
50 k2cw = k2c;
51 k2dw = [k2d zeros(uz)];

1.1.9.4 LQG Controller

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % LQG controller with integral action %
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4
5  % Augment plant with 2 integrators
6      ny      = size(C,1);
7      [nx,nu] = size(B); % n=5, m=2
8      Zxx     = zeros(nx,nx); Zxu = zeros(nx,nu);
9      Zux     = zeros(nu,nx); Zuu = zeros(nu,nu);
10     Aint    = [A Zxu; -C Zuu]; % Augment plant with integrators
11     Bint    = [B; -D];        % Augment plant with integrators (at
                                plant output)
12
13
14     states = 5;                % use old system
15                                     % -> old number of states
16
17 % Kalman filter (only for true states -
18 %                not needed for integrators)
19
20 Q      = [kw*eye(2)];
21 R      = kv*eye(2);
22 sys_d  = ss(A, [B Bd], C, [D D]); % D is zero anyway
23 [kestold,KFL,P,M,Z] = kalman(sys_d,Q,R);
24
25 % LQR control
26 states = 7;
27 Q      = 5*eye(states);
28 R      = 1*eye(2);
29 N      = 0;
30 [lqrK,S,e] = lqr(Aint,Bint,Q,R); % Shows that we only need
31                                     % the A and B matrices
32                                     % for LQR control
33
34 lqrKs = lqrK(:,1:nx);            % Plant state feedback part
35 lqrKint = lqrK(:,nx+1:nx+ny); % Integrator state feedback part

```

```

36 lqrKii = inv(lqrKint);
37
38 Aki = zeros(2,2); %State space representation of two
    integrators in parallel...
39 Bki = eye(2);
40 Cki = eye(2);
41 Dki = 1e-3*eye(2); %... with a small non-singular D term to
    make the Hanus form implementable
42
43 Ahki = Aki-Bki*inv(Dki)*Cki; %Two integrators in parallel in
    Hanus' form
44 Bhki = [zeros(ny,ny) Bki*inv(Dki)];
45 Chki = Cki;
46 Dhki = [Dki zeros(ny,nu)];
47
48 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
49 % Fast low-pass dynamics to break algebraic loop in Simulink
50 Af = -1000*eye(2);
51 Bf = eye(2);
52 Cf = 1000*eye(2);
53 Df = zeros(2,2);

```

## 1.2 Anti-windup with PI Controllers and Selectors

Selectors and 'overrides' are often used in plants when operational objectives change with operating conditions. Clearly, selectors makes the control system non-linear, and mistakes are easily made when designing such control systems, leading to sustained oscillations or even instability. It may reasonably be argued that more advanced control techniques, specifically MPC, can be a good alternative to regulatory control systems in which standard single-loop controllers (i.e., PI) are combined with selectors, since changes in the set of active constraints are handled relatively easily with MPC.

This assignment will address a particular problem of windup when using selectors. This windup is not caused by constraints in the manipulated variable, but occurs for inactive ('deselected') controllers, due to the absence of feedback when the controller is not selected. In this assignment a simple remedy to this problem is tested.

### 1.2.1 Introduction to Selective Control

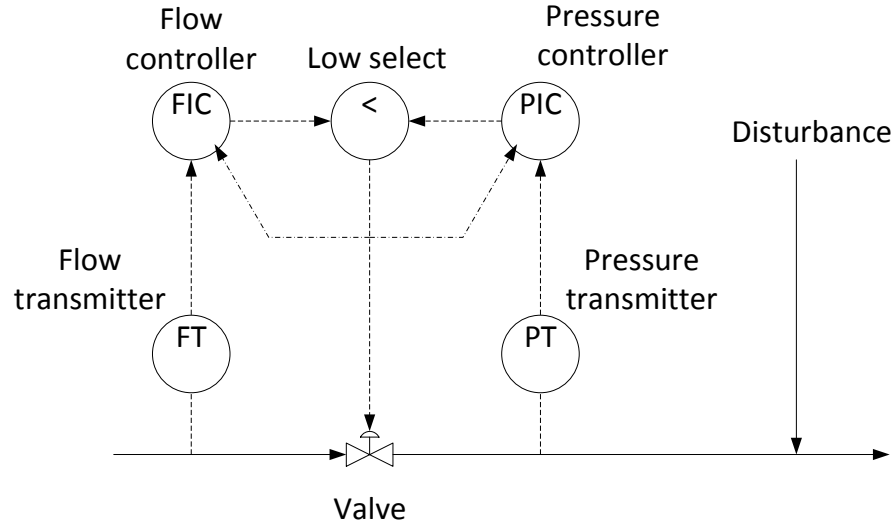
Selective control is sometimes used when there are more than one candidate controlled variable for a manipulated variable. For each of the candidate controlled variables a separate controller is the used, and the value of the

manipulated variable that is implemented is selected among the controller outputs. A simple example of selective control with pressure control on one side and flow control on the other side of a valve is shown in Fig. 1.25. Normally one selects simply the highest or lowest value. A few points should be made about this control structure:

- Clearly, a single manipulated variable can control only one controlled variable at the time, i.e., the only variable that is controlled at any instant is the variable for which the corresponding controller output is implemented. It might appear strange to point out such a triviality, but discussions with several otherwise sensible engineers show that many have difficulty comprehending this. Thus, one should consider with some care how such a control structure will work.
- The selection of the active controller is usually based on the controller outputs, not the controller inputs. Nevertheless the local operators and engineers often believe that the selection is based on the controller inputs, or that “the control switches when the a measurement passes its setpoint”. In principle, the selection of the active controller may also be based on the controller inputs<sup>1</sup>. Some type of scaling will then often be necessary, in order to compare different types of physical quantities (e.g., comparing flowrates and pressures).
- If the controllers contain integral action, a severe problem that is similar to “reset windup” can occur unless special precautions are taken. The controllers that are *not* selected, should be reset (for normal PID controller this is done by adjusting the value of the controller integral) such that for the present controller measurement, the presently selected manipulated variable value is obtained. Commonly used terms for this type of functionality are “putting the inactive controllers in tracking mode” or “using a feedback relay”. This functionality should be implemented with some care, as faulty implementations which permanently lock the inactive controllers are known to have been used. On a digital control system, the controllers should do the following for each sample interval:
  1. Read in the process measurement.
  2. Calculate new controller output.
  3. The selector now selects the controller output to be implemented on the manipulated variable.
  4. The controllers read in the implemented manipulated variable value.
  5. If the implemented manipulated variable value is different from the controller output, the internal variables in the controller (typically the integral value) should be adjusted to obtain the currently implemented manipulated variable value as controller output, for the current process measurement.

---

<sup>1</sup> Provided appropriate scaling of variables is used, the auctioneering control structure may be a better alternative to using selective control with the selection based on controller inputs.



**Fig. 1.25** Schematic of simple plant with two controllers and a selector. Note that the applied control signal is fed back to the controllers.

For PI controllers, the simple PI anti-windup scheme in Fig. 1.1 may be used.

### 1.2.2 The Control Problem in the Assignment

Consider the small plant depicted in Fig. 1.25. In normal operation, the flow controller should be active, but the pressure controller should take over if the downstream pressure becomes too high. This is achieved by selecting the lower of the two controller outputs, and applying that as the valve opening. A downstream disturbance affects both pressure and flow, and can cause the pressure control to take priority over the flow control.

In Fig. 1.25, solid arrows indicate process flows (in pipelines), dashed lines indicate signal transmission, and the dash-dotted lines indicate 'feedback' from the signal actually applied to the valve back to the controllers, for avoiding windup. In this assignment we will only study the effects of the selector, but one should note that there is little additional complexity involved in also having limited range of actuation of the valve, e.g., to the range  $\pm 1$ .

A simple model of the plant is given as

$$\begin{bmatrix} f(s) \\ p(s) \end{bmatrix} = \begin{bmatrix} 5 \\ 10s+1 \\ 10 \\ 100s+1 \end{bmatrix} u(s) + \begin{bmatrix} -1 \\ 20s+1 \\ 2 \\ 20s+1 \end{bmatrix} d(s) \quad (1.11)$$

where  $f$  is the flowrate,  $p$  is the pressure,  $u$  is the valve position, and  $d$  is the disturbance. Note that these variables, as always when using transfer function models, are expressed in *deviation variables*, and thus both negative valve positions, pressures and flowrates *do* make sense<sup>2</sup>.

### 1.2.3 Assignment

#### 1. Model implementation

Implement the a model of the plant in Simulink. The two controllers may both be given the setpoint zero. The disturbance should be modelled as a square wave with period 500.

#### 2. Controller tuning

Tune each of the PI controllers, without taking saturation or the selector into account. Any tuning methodology could be used (this should not be very difficult anyway, since the plant in each of the loops is Strictly Positive Real). However, completely unrealistic tunings should be avoided (i.e., for asymptotically stable plants like in this example, the closed loop time constant should not be orders of magnitude faster than the open loop time constant).

#### 3. Simulation without anti-windup

Implement the resulting PI controllers in Simulink – without accounting for windup, and simulate with the disturbance active and both controllers operating (i.e., with the selector).

Comment on the results of the simulation.

#### 4. Simulation with anti-windup

Implement the PI controllers with anti-windup, and redo the simulation. Comment on the results and compare to the simulation results without anti-windup.

### 1.2.4 Solution

#### 1.2.4.1 Model Implementation

The Simulink implementation of the plant model, with the disturbance, and including PI controllers (without anti-windup) and selector, is shown in Fig. 1.26

---

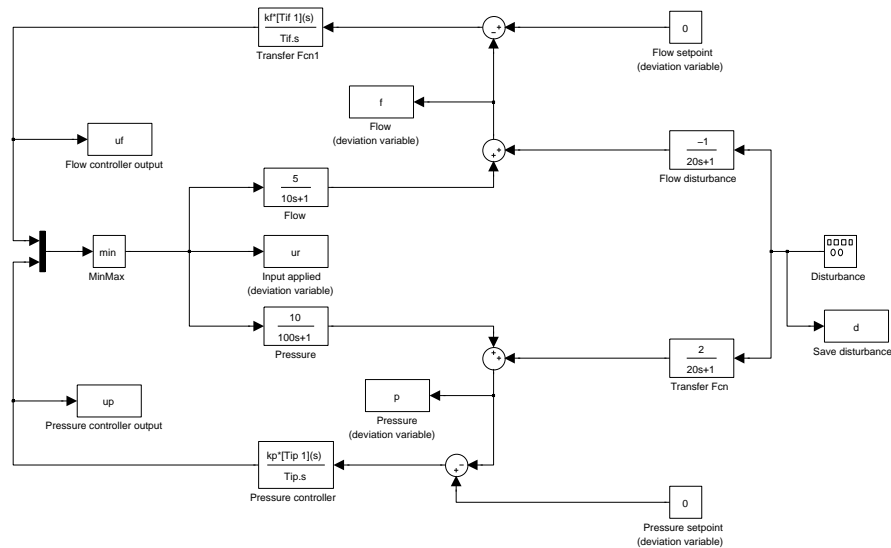
<sup>2</sup> In addition, the variables are scaled, so not too much emphasis should be placed on the magnitude of the variables

### 1.2.4.2 Controller Tuning

Not much effort has been put into the controller tuning. For each of the controllers, the integral time  $T_i$  has been set equal to the (dominant) time constant of the open loop, and a proportional gain of 1 is used. These tuning parameters are found to result in reasonable responses to setpoint changes for each individual loop.

### 1.2.4.3 Simulation without Anti-windup

The system has been simulated with the disturbance active. The response in the controlled outputs are shown in Fig. 1.27. We observe large variations in the controlled variables, including variations where the measurement is larger than the setpoint (0 for both loops). The corresponding controller outputs are shown in Fig. 1.28. We observe that the pressure controller winds up so severely, that it needs a long time to wind down again, and therefore only is selected (active) for a short period during each disturbance cycle. This explains the long periods where the pressure measurement is above the setpoint. There are also large deviations in the flow measurement, but since the flow controller is selected most of the time, the deviations in the flow measurement are quite quickly removed.



**Fig. 1.26** Simulink implementation of model and control system, without anti-windup

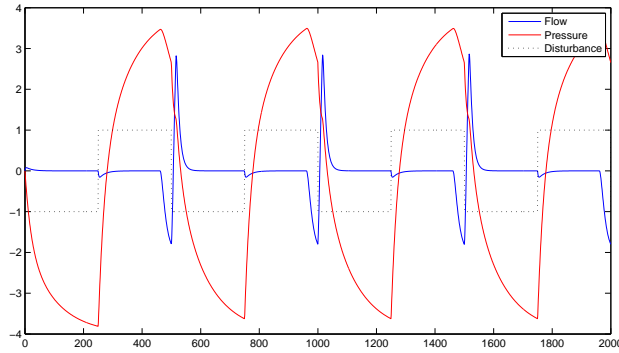


Fig. 1.27 Responses in controlled variables, without anti-windup.

#### 1.2.4.4 Simulation with Anti-windup

The simple anti-windup technique for PI controllers described in [Hovd and Sivalingam \(2011\)](#) is used, and is implemented as illustrated in [Fig. 1.29](#). Note that if we want to account for saturation in the valve in addition to the effect of the selector, this could easily be done in the simulation by putting a saturation element in series after the selector, before sending the 'anti-windup signal' to each controller. In a real life implementation, it would be preferable to use for anti-windup a direct measurement of the valve position.

The simulated responses in the controlled variables, when including anti-windup, are shown in [Fig. 1.30](#), and the corresponding responses in the controller outputs are shown in [Fig. 1.31](#). We see that only small deviations of

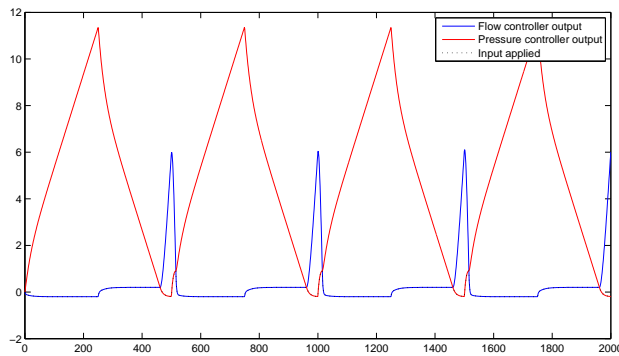
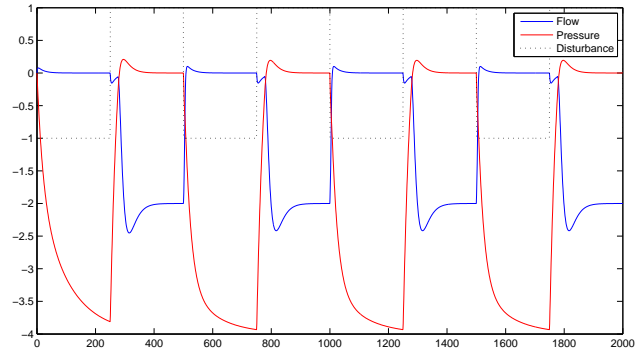


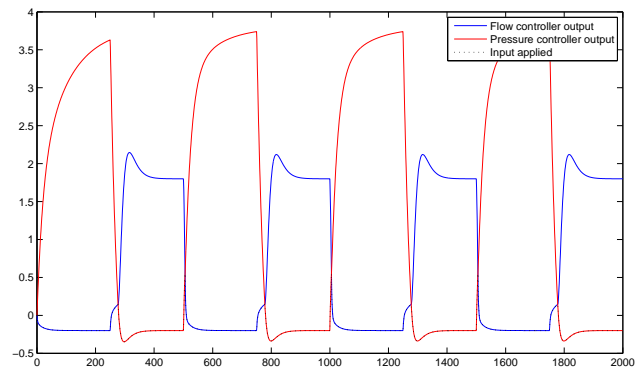
Fig. 1.28 Responses in controller outputs, without anti-windup.







**Fig. 1.30** Responses in controlled variables, with anti-windup.



**Fig. 1.31** Responses in controller outputs, with anti-windup.

### 1.3 Stiction Detection

In process control, valves are commonly used as manipulated variables. Stiction in control valves is a common cause for poor control performance, often causing continuous oscillations in the loop. This assignment will study the detection of valve stiction, using a few of the available stiction detection methods. Background on stiction detection methods can be found in [Sivalingam and Hovd \(2011\)](#).

A Simulink model is used to simulate a simple control loop with a (possibly) sticking valve. A printout of the Simulink model is shown in [Fig. 1.32](#). The submodel `Intstiction` contains the integration from net force acting on the valve to the valve velocity, as well as the calculation of the friction

force, based on the simple model described in [Sivalingam and Hovd \(2011\)](#). The submodel is shown in Fig. 1.33. For simulations without stiction, the submodel `Intstiction` is replaced by the simple integration (from net force to velocity), with a positive  $k_d$  in Fig. 1.32 used to simulate viscous (linear) friction.

### 1.3.1 Assignment

Prepare by downloading required software (if not already installed) and the data series used in the assignment.

1. Download the Higher Order Spectral Analysis (hosa) toolbox from Matlab Central, and/or the file `bicoherence.m` from Dr. MAA Shoukat Choudhury's website [teacher.buet.ac.bd/shoukat/](http://teacher.buet.ac.bd/shoukat/).
2. Download the Matlab data files `Set1data.mat` – `Set4data.mat` from [http://www.itk.ntnu.no/ansatte/Hovd\\_Morten/NIL/](http://www.itk.ntnu.no/ansatte/Hovd_Morten/NIL/).

Each of the Matlab data files contain three arrays of variables: i)  $OP$ , the controller output, ii)  $PV$ , the process (measured) variable, and iii)  $tp$ , the corresponding time index.

For each of the data sets, do the following:

1. Plot both  $OP$  and  $PV$  versus time. Do these plots indicate oscillations caused by stiction?
2. Use the cross-correlation method to detect (possible) stiction. Comment on the result.
3. Use the  $OP - PV$  plot to detect stiction. If the plot is hard to interpret, try time-shifting the data series relative to each other. Comment on the result.

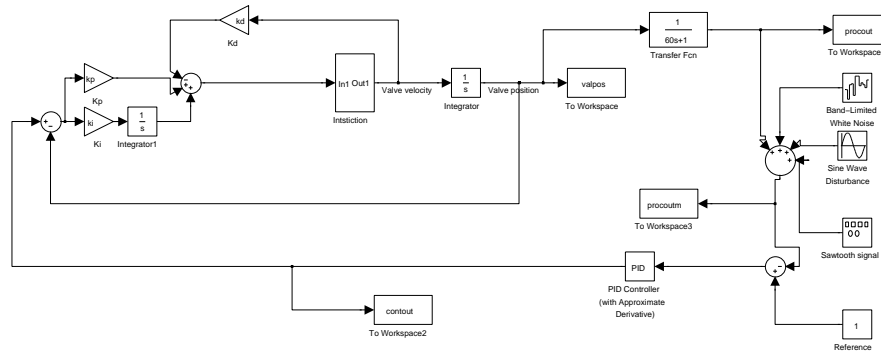
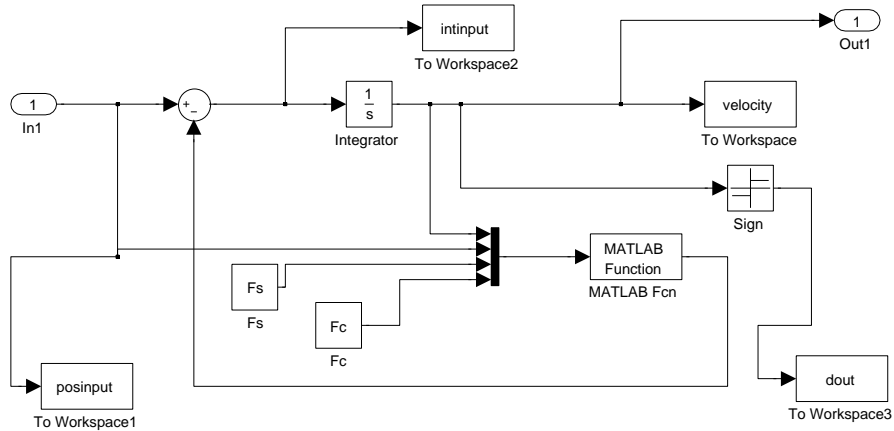


Fig. 1.32 Simulink model for simulating simple loop with sticky valve



**Fig. 1.33** Simulink submodel for calculating friction force and valve velocity

- Use `glstat.m` (from the `hosa` toolbox) and/or `bicoherence.m` to detect stiction. Comment on the result.

### 1.3.2 Solution

The data series contain an initial setpoint change (from 0 to 1). It may therefore be advisable to select parts of the data series after the *PV* has come close to the setpoint. The length of the selected data series should contain multiple oscillations. You may also use the Matlab function `detrend` to remove any linear trends in the data.

#### 1.3.2.1 Data Set 1

The time plots of *OP* and *PV* for data set 1 are shown in Fig. 1.34. The *OP* resembles the sawtooth shape, and *PV* resembles the square wave shape, that together are held to be typical of stiction.

The cross-correlation function for data set 1 is shown in Fig. 1.35. This is nearly an odd-valued function, and this test therefore indicates the presence of stiction.

The *OP* – *PV* plot for data set 1 is shown in Fig. 1.36, for approximately two oscillation periods. Despite the imperfect trend removal, the plot clearly shows the signals 'oscillating around an open area', and sharp corners in the plots. This test therefore also indicates the presence of stiction.

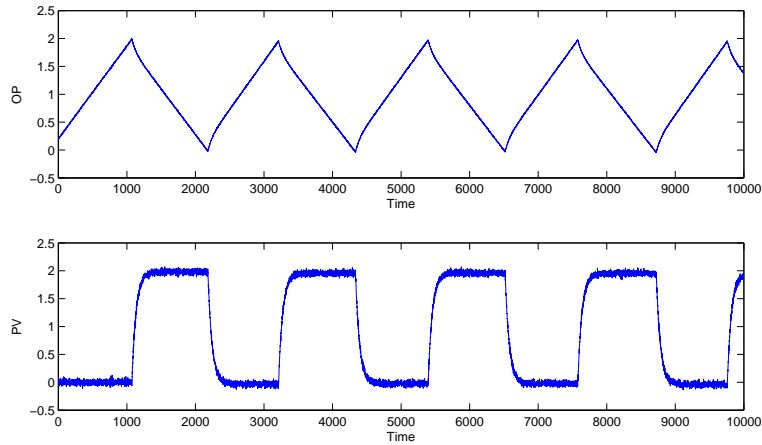


Fig. 1.34  $OP$  and  $PV$  for data set 1.

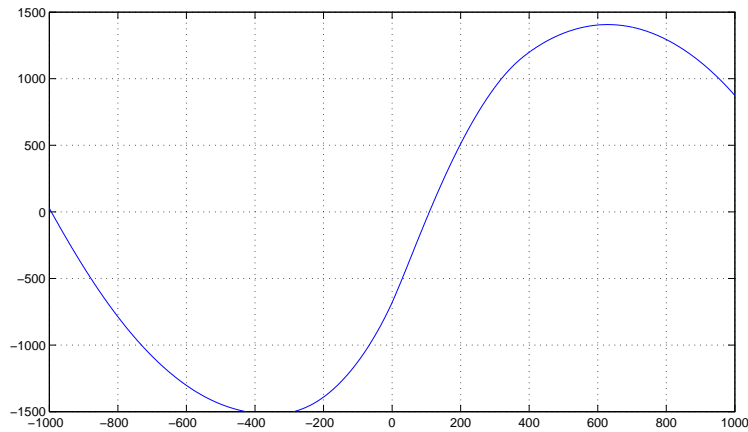
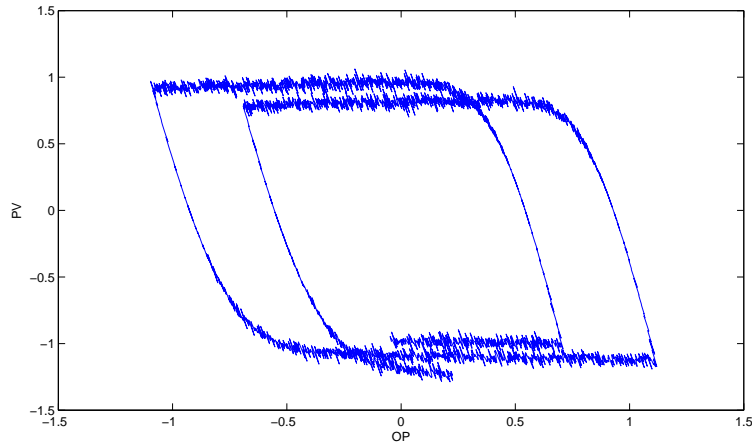


Fig. 1.35 The  $OP$  to  $PV$  cross-correlation function for data set 1.

The routines `bicoherence.m` and `glstat.m` are both run to detect the presence of nonlinearity in the output. Default parameters are used for both routines. Elements 5001-9096 of the  $PV$  data series are used (a data series of length a power of 2 is used since these routines are based on the FFT). The command

```
[bic,waxis,NGI,NLI,bic_f_max] = bicoherence(pv)
```

gives  $NGI = -0.0014$  and  $NLI = 0.4394$ . Since both these indices are re-



**Fig. 1.36** *OP-PV* plot for data set 1.

quired to be positive in order to indicate significant non-linearity, no stiction is indicated. In contrast, the command

```
[sg,sl]=glstat(pv,0.51,128)
```

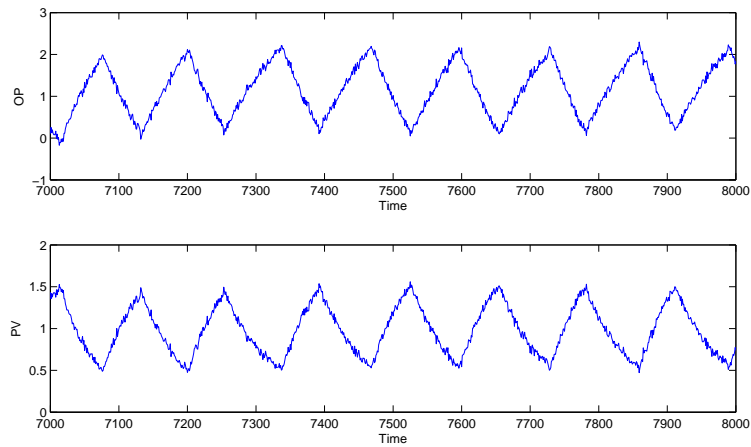
shows that the signal is non-Gaussian (the Pfa, probability of false alarm, is zero). However, the estimated inter-quartile range  $R$  is quite close to the theoretical value for a linear system, and thus `glstat` does not find any clear sign of non-linearity.

### 1.3.2.2 Data Set 2

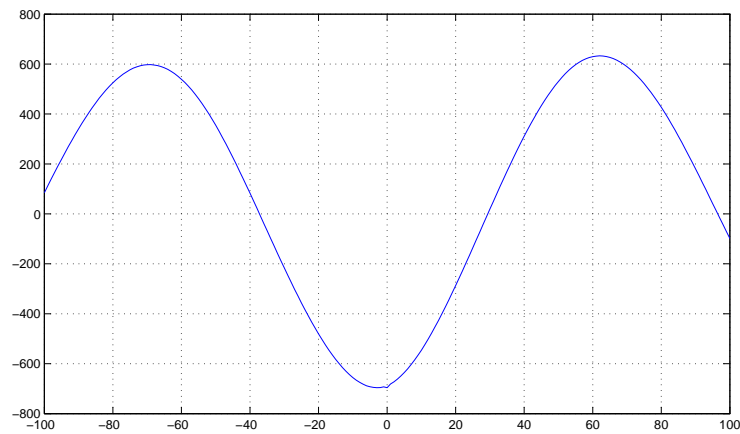
The time plots of *OP* and *PV* for data set 2 are shown in Fig. 1.37. The *OP* resembles the sawtooth shape, but here *PV* also resembles a sawtooth shape, and stiction is therefore not readily identifiable from the time plots.

The cross-correlation function for data set 2 is shown in Fig. 1.38. This is nearly an even-valued function, and this test therefore does not indicate the presence of stiction.

The *OP* – *PV* plot for data set 2 is shown in Fig. 1.39. The plot looks like an ellipsoid seen from the side, but due to the angle it is difficult to assess whether there are any sharp corners in the plot. In Fig. 1.40, the *PV* time series is therefore shifted by 30 samples. The plot is a little messy, since the time shifting destroys the correspondence between the effects of measurement noise in the *OP* and *PV* time series. However, the sharp corners in the plot are now clearly identifiable, indicating the presence of stiction.

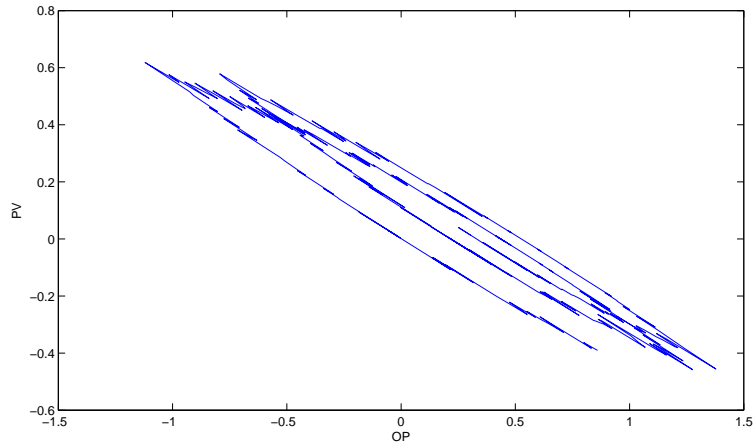


**Fig. 1.37** *OP* and *PV* for data set 2.

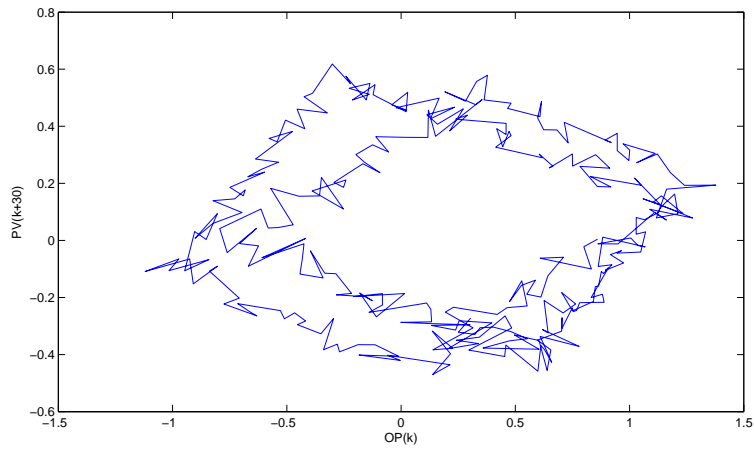


**Fig. 1.38** The *OP* to *PV* cross-correlation function for data set 2.

The routine `bicoherence.m` returns a negative value for *NGI* and a positive value for *NLI*, and therefore nonlinearity has not been detected for this *PV* time series either. The routine `glstat.m` finds no clear indication of *PV* being non-Gaussian, the probability of false alarm being estimated to 0.8665. None of these routines can therefore detect stiction in the loop.



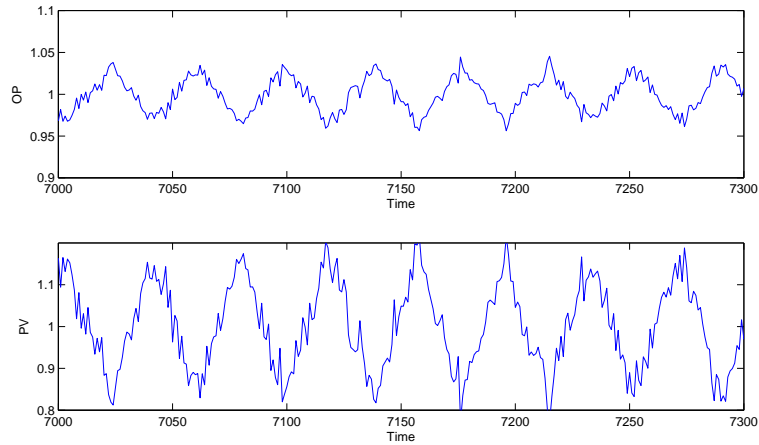
**Fig. 1.39** *OP-PV* plot for data set 2.



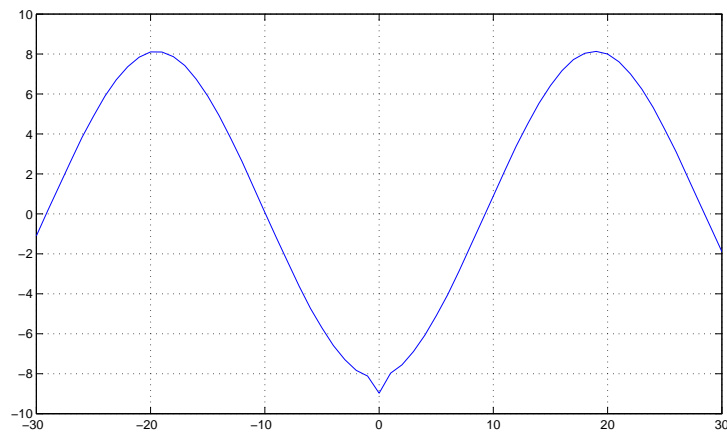
**Fig. 1.40** *OP-PV* plot for data set 2, with the *PV* data series shifted 30 samples.

### 1.3.2.3 Data Set 3

The time plots of *OP* and *PV* for data set 3 are shown in Fig. 1.41. The *OP* resembles the sawtooth shape, but here *PV* also resembles a sawtooth shape, and stiction is therefore not readily identifiable from the time plots. The oscillations are of modest amplitude (around an order of magnitude larger than the measurement noise).



**Fig. 1.41**  $OP$  and  $PV$  for data set 3.

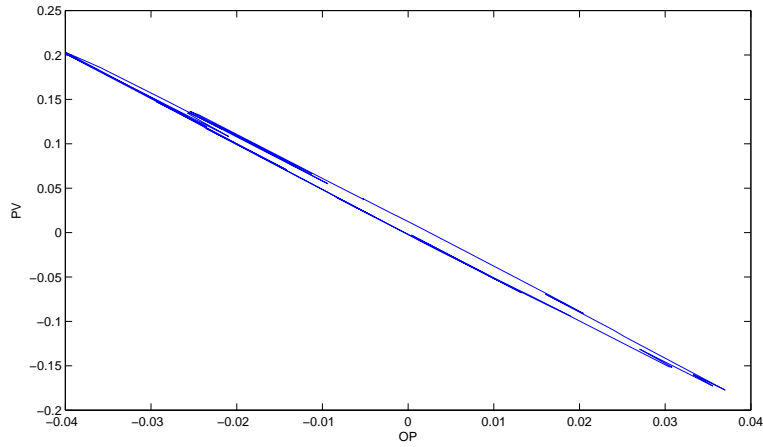


**Fig. 1.42** The  $OP$  to  $PV$  cross-correlation function for data set 3.

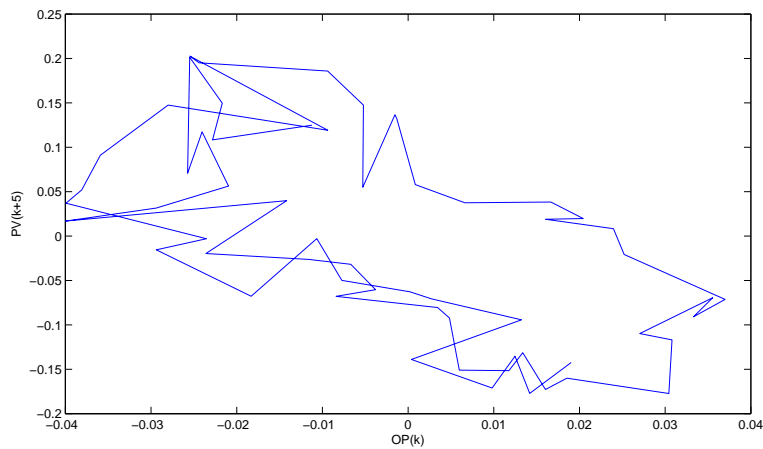
The cross-correlation function for data set 3 is shown in Fig. 1.42. This is an almost perfectly even-valued function, and this test therefore does not indicate the presence of stiction.

The  $OP - PV$  plot for data set 3 is shown in Fig. 1.43. The plot looks like an ellipsoid seen from the side, but due to the angle it is difficult to assess whether there are any sharp corners in the plot. In Fig. 1.44, the  $PV$  time series is therefore shifted by 5 samples. Again, we find that the plot is a





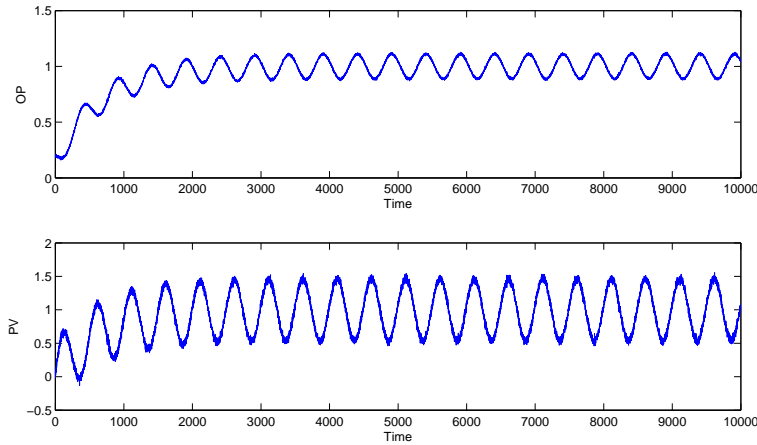
**Fig. 1.43** *OP-PV* plot for data set 3.



**Fig. 1.44** *OP-PV* plot for data set 3, with the *PV* data series shifted 5 samples.

little messy, since the time shifting destroys the correspondence between the effects of measurement noise in the *OP* and *PV* time series. Furthermore, since the measurement noise is here of significant magnitude compared to the amplitude of the oscillations, it is also in this plot not possible to identify any sharp corners in the plot.

The routine `bicoherence.m` returns a negative value for *NGI* and a positive value for *NLI*, and therefore nonlinearity has not been detected for this



**Fig. 1.45** *OP* and *PV* for data set 4.

*PV* time series either. Similarly, the routine `glstat.m` finds no clear indication of *PV* being non-Gaussian, the probability of false alarm being estimated to 1.0. None of these routines can therefore detect stiction in the loop.

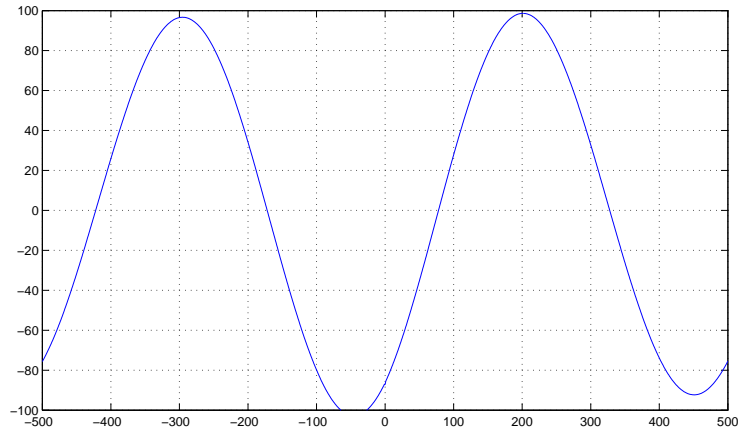
#### 1.3.2.4 Data Set 4

The time plots of *OP* and *PV* for data set 4 are shown in Fig. 1.45. Here both *OP* and *PV* look like noisy sinusoidal signals, without any clear indication of stiction.

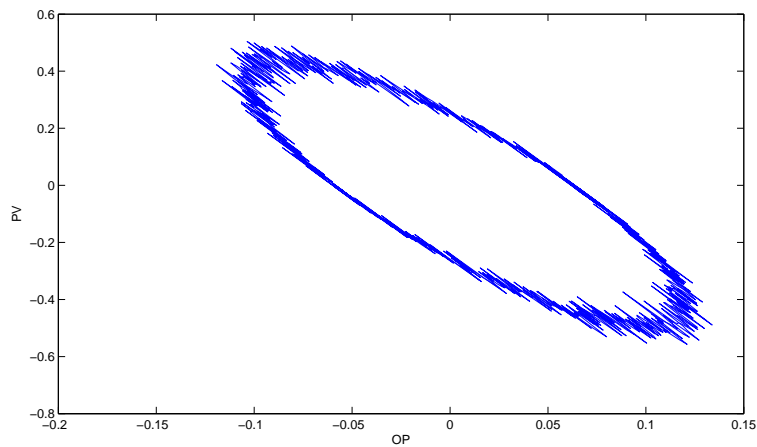
The cross-correlation function for data set 4 is shown in Fig. 1.46. This is nearly an even-valued function, and this test therefore does not indicate the presence of stiction.

The *OP* – *PV* plot for data set 3 is shown in Fig. 1.43. The plot looks like a smooth ellipsoid (when disregarding the obvious effects of noise), and therefore does not indicate the presence of stiction.

Again the routine `bicoherence.m` returns a negative value for *NGI* and a positive value for *NLI*, and therefore nonlinearity has not been detected for this *PV* time series. Similarly, the routine `glstat.m` finds no clear indication of *PV* being non-Gaussian, the probability of false alarm being estimated to 1.0. None of these routines therefore indicate stiction in the loop.



**Fig. 1.46** The *OP* to *PV* cross-correlation function for data set 4.



**Fig. 1.47** *OP-PV* plot for data set 4.

### 1.3.3 Conclusions

1. Data set 1 is a 'traditional' stiction example, as seen from Fig. 1.34. The oscillations are here at a significantly lower frequency than the dominant dynamics of the plant.
2. Data set 2 is obtained with the same stiction model parameters, but with a higher proportional gain in the controller. The oscillations are therefore

faster – in a similar frequency range as the dominant dynamics of the plant.

3. Data set 3 is also obtained with the same stiction model parameters. However, here integral action is used in the valve positioner. This results in rather fast oscillations – so fast that the oscillations in the output are small due to the low plant gain at high frequencies. Note, however, that the oscillations in the valve position due to the stick jump are nevertheless significant in magnitude, and may result in premature wear on the valve.
4. Data set 4 is obtained with a sinusoidal external disturbance, and the control loop itself is linear

We find that the cross-correlation method therefore perform well on data sets 1 and 4, and the OP-PV plot (at least when allowing for time-shifting between the OP and PV data series) perform well on data sets 1, 2, and 4. The methods using only the PV (`bicoherence.m` and `glstat.m`) failed to detect significant nonlinearity for sets 1, 2, and 3, while correctly finding no indication of nonlinearity for data set 4. One should note, however, that

- It is not really fair to compare a stiction detection method using only PV to methods using both OP and PV.
- Both `bicoherence.m` and `glstat.m` have tunable parameters that may help in detecting nonlinearity in some (or all?) of these cases. However, in the absence of any clear recommendations on how to select these parameters, we have chosen to use default values throughout.

None of the methods tested were able to detect stiction in data set 3. Logging the actual valve position (which must be measured – at least locally – in order to implement a valve positioner) would be very helpful for detection of stiction when the stiction (in combination with the valve positioner) results in such fast oscillations.

## 1.4 Controller Performance Monitoring using the Harris Index

In large-scale chemical plants, the number of control loops is simply too high for operators or engineers to continuously monitor the performance of each loop. Studies show that disappointingly many control loops perform poorly. The need for automated tools to assess control performance is therefore clear. In this assignment, one of the most popular performance monitoring methods, the Harris Index, will be illustrated. More background on the Harris Index and its modifications can be found in [Sivalingam and Hovd \(2011\)](#).

In this assignment the Harris Index is used to assess the performance of a simple control loop. The sampling interval of the loop is 1 (time unit) and the open-loop dominant time constant is estimated to be around 100 (the

knowledge of the dominant time constant is not known for calculating the Harris Index, but is used in one of the more common modifications of the index). The plant is strictly proper, but there is otherwise no time delay from input to output, i.e.,  $h(0) = 0$ ,  $h(1) \neq 0$  where  $h(k)$  is the impulse response coefficient for lag  $k$ . The plant is open loop stable and has no zeros outside the unit disk.

### 1.4.1 Assignment

1. Download the Matlab data file `cpmsim.mat` from [http://www.itk.ntnu.no/ansatte/Hovd\\_Morten/NIL/](http://www.itk.ntnu.no/ansatte/Hovd_Morten/NIL/). The file contains the measurement (OP) of a control loop. Since the setpoint is 0, the measurement and control offset are equivalent.
2. Calculate the Harris Index (or Normalized Harris Index) for the control loop. Is there scope for significant improvements in control performance?
3. Select a reasonable value for the desired closed loop dominant time constant. Instead of the minimum variance benchmark used in the Harris Index, use a modified benchmark reflecting a desired closed loop time constant to assess control performance.

The calculations involved are relatively simple, and appropriate Matlab routines may be programmed by the student. Alternatively, the Matlab routines `ar` and `impulse` may be useful.

### 1.4.2 Solution

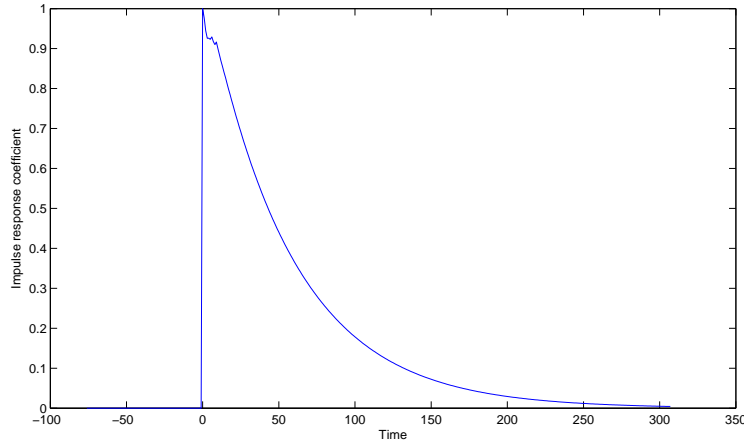
First the mean of the data series OP is removed with the command `opd = detrend(op, 'constant')`. Next, a 10th-order auto-regressive model from the (unmeasured) disturbance to the controlled variable is identified using the command `mod=ar(opd,10)`. The routine `ar()` finds parameters for the model

$$y(k) + \alpha_1 y(k-1) + \alpha_2 y(k-2) + \dots + \alpha_{10} y(k-10) = a(k) \quad (1.12)$$

$$\Downarrow$$

$$A(q^{-1})y(k) = d(k)$$

where  $y(k)$  is the measurement at time  $k$ , and  $a(k)$  is the (unmeasured) disturbance. Note that the leading coefficient of the polynomial  $A(q^{-1})$ ,  $\alpha(0) = 1$ . The coefficients of the polynomial  $A(q^{-1})$  are found in `mod.a`, and the estimated variance of  $a(k)$  is found in `mod.NoiseVariance`. The AR model is converted to an impulse response model by simple polynomial long divi-



**Fig. 1.48** Calculated impulse response

sion, or using the command `[h,t] = impulse(mod)`. The resulting impulse response is plotted in Fig. 1.48.

It can be seen that the model is (as expected) semi-proper. There is no time delay from manipulated variable to the controlled variable, so the minimum variance can be found from

$$\sigma_{mv}^2 = h(0)^2 \sigma_a^2 \quad (1.13)$$

We thus find  $\sigma_{mv}^2 = 2.489 \cdot 10^{-4}$ . The actual variance of the output is found to be  $\sigma_y^2 = 7.95 \cdot 10^{-3}$ . We thus find a Harris index of  $HI = 31.94$  or a normalized Harris Index of  $NHI = 0.969$ . We conclude that there is a significant scope for improved control performance.

A frequently quoted rule of thumb is that the closed loop time constant should be 2-3 time faster than the open loop time constant. However, this clearly cannot apply to integrating processes (with infinite time constant), and in practice will depend on the available range and speed of actuation for the manipulated variable. Nevertheless, the rule of thumb would indicate that a closed loop time constant of 40 is reasonable, considering the open loop dominant time constant of 100. This corresponds to a desired closed loop (discrete time) pole of  $\mu = e^{-1/40} = 0.9753$ . The modified benchmark variance therefore becomes

$$\sigma_{mod}^2 = \left(1 + \frac{\mu^2}{1 - \mu^2}\right) \sigma_a^2 = 5.103 \cdot 10^{-3} \quad (1.14)$$

With this modified benchmark variance, we find a modified Harris index of  $HI_{mod} = 1.49$ , or  $NHI_{mod} = 0.327$ . Thus, with the chosen closed loop time constant as basis for comparison, the motivation for retuning the controller is significantly reduced, as the observed variance is not much larger than the 'acceptable' variance.

**Acknowledgements** The authors are pleased to acknowledge the financial support by a grant No. NIL-I-007-d from Iceland, Liechtenstein and Norway through the EEA Financial Mechanism and the Norwegian Financial Mechanism.

The contributions of Anders Fougner and Anders Willersrud in preparing the anti-windup assignment are also gratefully acknowledged.

## References

- Hanus R, Kinnaert M, Henrotte JL (1987) Conditioning technique, a general anti-windup and bumpless transfer method. *Automatica* 23(6):729–739
- Hovd M, Sivalingam S (2011) A short introduction to anti-windup. In: Huba M, Skogestad S, Fikar M, Hovd M, Johansen TA, Rohal'-Ilkiv B (eds) *Selected Topics on Constrained and Nonlinear Control*. Textbook, STU Bratislava – NTNU Trondheim
- Sivalingam S, Hovd M (2011) Controller performance monitoring and assessment. In: Huba M, Skogestad S, Fikar M, Hovd M, Johansen TA, Rohal'-Ilkiv B (eds) *Selected Topics on Constrained and Nonlinear Control*. Textbook, STU Bratislava – NTNU Trondheim
- Skogestad S, Postlethwaite I (2005) *Multivariable Feedback Control. Analysis and Design*. John Wiley & Sons Ltd, Chichester, England
- Wade HL (1997) Inverted decoupling – a neglected technique. *ISA Transactions* 36:3–10

*Comments – Remarks*



*Comments – Remarks*

*Comments – Remarks*

*Comments – Remarks*

## Chapter 2

# Optimal Use of Measurements for Control, Optimization and Estimation using the Loss Method: Summary of Existing Results and Some New

Sigurd Skogestad and Ramprasad Yelchuru and Johannes Jäschke

**Abstract** The objective of this chapter is to study the optimal use of measurements and measurements combinations,  $\mathbf{c} = \mathbf{H}\mathbf{y}$  in optimization and estimation based on the loss method.

### 2.1 Introduction

In this paper we consider a (steady-state) unconstrained quadratic optimization problem with linear measurement relationships. The main objective is to find a linear measurement combination,  $\mathbf{c} = \mathbf{H}\mathbf{y}$ , such that control of these indirectly leads to close-to-optimal operation with a small loss  $L$ , in spite of unknown disturbances,  $\mathbf{d}$ , and measurement noise (error),  $\mathbf{n}^y$ . If the original optimization problem is constrained, then we assume that any active constraints are kept constant (controlled) and we consider the lower-dimensional unconstrained subspace. Depending on the disturbance range considered, there may be several constrained regions, and the procedure of finding  $\mathbf{H}$  needs to be repeated in each constrained region. Switching between the regions will then be needed, and we will show that monitoring the controlled variables  $\mathbf{c} = \mathbf{H}\mathbf{y}$  in neighboring regions can be used for switching.

---

Sigurd Skogestad

Department of Chemical Engineering, Norwegian University of Science and Technology in Trondheim, Norway, e-mail: [skoge@chemeng.ntnu.no](mailto:skoge@chemeng.ntnu.no)

Ramprasad Yelchuru

Department of Chemical Engineering, Norwegian University of Science and Technology in Trondheim, Norway, e-mail: [ramprasad.yelchuru@chemeng.ntnu.no](mailto:ramprasad.yelchuru@chemeng.ntnu.no)

Johannes Jäschke

Department of Chemical Engineering, Norwegian University of Science and Technology in Trondheim, Norway, e-mail: [jaschke@chemeng.ntnu.no](mailto:jaschke@chemeng.ntnu.no)

What we here call the “loss method” is the same as what is called the “exact local method” in these papers (Halvorsen et al (2003); Alstad et al (2009)).

The new material in this summary paper is mainly related to using data ( $\mathbf{Y}, \mathbf{X}$ ) as the basis, and for example, to use the “loss” method for regression, see section 5, data approach 1 and 2.

## 2.2 Problem Formulation

### 2.2.1 Classification of Variables

- $\mathbf{u}$  - inputs (degrees of freedom) for optimization and control (it does not actually matter what they are as long as they form an independent set)
- $\mathbf{d}$  - disturbances, including parameter changes.
- $\mathbf{y}$  - all available measurements (will later call a subset of these for  $\mathbf{x}$  in accordance with statistics notation). The manipulated variables (MVs, often the same as the inputs  $\mathbf{u}$ ) are generally included in the measurement set  $\mathbf{y}$ . This will allow, for example, for simple control policies where the inputs are kept constant. Of course, the set  $\mathbf{y}$  also includes measured disturbances ( $\mathbf{d}_m$ , a subset of  $\mathbf{d}$ ).
- $\mathbf{n}^y$  - measurement noise (error) for  $\mathbf{y}$ ,  $\mathbf{y}_m = \mathbf{y} + \mathbf{n}^y$ .
- $\mathbf{p}$  - prices = weights that enter into cost function (do not affect  $\mathbf{y}$ )

### 2.2.2 Cost Function

The objective is to choose the input  $\mathbf{u}$  to minimize the quadratic cost function

$$\mathbf{J}(\mathbf{u}, \mathbf{d}) = \mathbf{u}^T \mathbf{Q}_1 \mathbf{u} + \mathbf{d}^T \mathbf{Q}_2 \mathbf{d} + \mathbf{u}^T \mathbf{Q}_3 \mathbf{d} \quad (2.1)$$

Note that for simplicity, we have not included linear terms in the cost function. Any linear term in  $\mathbf{u}$  can be removed by shifting the zero point for  $\mathbf{u}$  to be at the optimal point. On the other hand, a linear term in  $\mathbf{d}$  cannot be counteracted by choosing the input  $\mathbf{u}$ , so excluding it does not change the solution. The same applies to any constant term in the cost.

If we compare (2.1), with a second-order Taylor series expansion of the cost around a nominal point  $(\mathbf{u}^*, \mathbf{d}^*)$ , then we have that

$$\mathbf{Q}_1 = \frac{1}{2} \mathbf{J}_{uu}^*, \mathbf{Q}_2 = \mathbf{J}_{ud}^*, \mathbf{Q}_3 = \frac{1}{2} \mathbf{J}_{dd}^*$$

and  $\mathbf{u}$  represents deviation from the optimal point  $(\mathbf{u}^*, \mathbf{d}^*) = (0, 0)$  at which  $\mathbf{J}_u^* = 0$ .

### 2.2.3 Measurement Model

A linear model is assumed for the effect on  $\mathbf{u}$  and  $\mathbf{d}$  on measurements  $\mathbf{y}$  (deviation variables)

$$\mathbf{y} = \mathbf{G}^y \mathbf{u} + \mathbf{G}_d^y \mathbf{d} = \tilde{\mathbf{G}} \begin{bmatrix} \mathbf{u} \\ \mathbf{d} \end{bmatrix} \quad (2.2)$$

### 2.2.4 Assumptions

- No constraints ( $\mathbf{u}$  spans unconstrained subspace)
- We want to find as many controlled variables as there are degrees of freedom,  $n_c = \dim(\mathbf{c}) = \dim(\mathbf{u}) = n_u$ . Then  $\mathbf{H}\mathbf{G}^y$  is a square  $n_u \times n_u$  matrix
- We use at least as many measurements as there are degrees of freedom,  $n_y \geq n_u = n_c$ .

### 2.2.5 Expected Set of Disturbances and Noise

We write  $\mathbf{d} = \mathbf{W}_d \mathbf{d}'$  where  $\mathbf{W}_d$  is a diagonal matrix giving the expected magnitude of each disturbance and  $\mathbf{d}'$  is a normalization vector of unit magnitude.

Similarly,  $\mathbf{n}^y = \mathbf{W}_{n^y} \mathbf{n}^{y'}$  where  $\mathbf{W}_{n^y}$  is a diagonal matrix with the magnitude of the noise for each measurement, and the vector  $\mathbf{n}^{y'}$  is of unit magnitude.

More precisely, the combined normalization vectors for disturbances and measurement noise are assumed to have 2-norm less than 1,

$$\left\| \begin{bmatrix} \mathbf{d}' \\ \mathbf{n}^{y'} \end{bmatrix} \right\|_2 \leq 1 \quad (2.3)$$

The choice of the 2-norm (rather than, for example, the vector infinity-norm) is discussed in the Appendix of [Halvorsen et al \(2003\)](#).

### 2.2.6 Problem

Given that

$$\mathbf{H} \underbrace{(\mathbf{y} + \mathbf{n}^y)}_{\mathbf{y}_m} = \mathbf{c}_s \quad (\text{constant} = 0 \text{ nominally}) \quad (2.4)$$

find the optimal  $\mathbf{H}$  such that “magnitude” of the loss

$$L = \mathbf{J}(\mathbf{u}, \mathbf{d}) - \mathbf{J}_{opt}(\mathbf{d}) \quad (2.5)$$

is minimized for the “expected”  $\mathbf{d}$  and  $\mathbf{n}^y$ .

The “expected” set of the disturbances and noise is defined above.

The “magnitude” of the loss still needs to be defined. Two possibilities are considered.

- Worst-case loss,  $L_{wc}$ .
- Average loss,  $L_{avg}$ .

### 2.2.7 Examples of this Problem

1. Identify controlled variables,  $\mathbf{c} = \mathbf{H}\mathbf{y}$  (“squaring down”). Then use feedback control to adjust  $\mathbf{u}$  such that  $\mathbf{c}_m = \mathbf{H}\mathbf{y}_m = \mathbf{c}_s$ .
2. Find invariants for quadratic optimization problems.
3. Obtain estimate of primary variables,  $\mathbf{c} = \hat{\mathbf{y}}_1 = \mathbf{H}\mathbf{y}$ .

Problem: Given that  $\hat{\mathbf{y}}_1 = \mathbf{H}\mathbf{y}$  find optimal  $\mathbf{H}$  such that magnitude of  $\|\mathbf{y}_1 - \hat{\mathbf{y}}_1\|$  is minimized for the expected  $\mathbf{d}$ 's and  $\mathbf{n}^y$ 's.

### 2.2.8 Comments on the Problem

1. The controlled variables are  $\mathbf{c} = \mathbf{H}\mathbf{y}$  and the objective is to find the non-square  $n_c \times n_y$  matrix  $\mathbf{H}$  (note that  $n_c = n_u$ ). In general,  $\mathbf{H}$  is a “full” combination matrix. However, it may also be interesting to consider control of individual measurements, in which case  $\mathbf{H}$  is a “selection” matrix with  $n_u$  number of columns with single 1 and the rest of columns are zero (mathematically  $\mathbf{H}\mathbf{H}^T = \mathbf{I}$ ).
2. Minimizing (the magnitude of) the loss  $L$  is close to but not quite the same as minimizing the cost  $J$ . In some cases they give identical results in terms of the optimal  $\mathbf{H}$ , for example, if we consider the average loss or cost for given disturbances (because then the same cost function is subtracted). So it seems it is the same for the 2-norm (Frobenius) of  $\mathbf{M}$  (see below). However, there will be some difference if we consider the worst-case loss or cost.

### 2.3 Solution to Problem: Preliminaries

The objective is to derive the solution to the above problem. It has been previously known as the “exact local method”, but we will here call it the loss method. However, first we need some preliminaries

#### 2.3.1 Expression for $\mathbf{u}_{opt}(\mathbf{d})$

We want to find the optimal input  $\mathbf{u}$  for a given disturbance  $\mathbf{d}$ . Expanding the gradient  $\mathbf{J}_u$  around the nominal point  $(\mathbf{u}^*, \mathbf{d}^*) = (0, 0)$  gives

$$\mathbf{J}_u = \mathbf{J}_u^* + \mathbf{J}_{uu}^* \mathbf{u} + \mathbf{J}_{ud}^* \mathbf{d} = \underbrace{\mathbf{J}_u^*}_{=0} + [\mathbf{J}_{uu}^* \quad \mathbf{J}_{ud}^*] \begin{bmatrix} \mathbf{u} \\ \mathbf{d} \end{bmatrix}$$

where  $\mathbf{J}_u^* = \mathbf{J}_u(\mathbf{u}^*, \mathbf{d}^*) = 0$  because the nominal point is assumed to be optimal. To remain optimal,  $\mathbf{u} = \mathbf{u}_{opt}(\mathbf{d})$ , we must have  $\mathbf{J}_u = 0$  and we derive

$$\mathbf{u}_{opt} = -\mathbf{J}_{uu}^{-1} \mathbf{J}_{ud} \mathbf{d} \quad (2.6)$$

where we have dropped the superscript  $*$  (either because we consider small deviations or because we assume that the problem is truly quadratic).

#### 2.3.2 Expression for $J$ around $\mathbf{u}_{opt}(\mathbf{d})$

Consider a given disturbance  $\mathbf{d}$ . Then expanding the cost  $J$  around a “moving”  $\mathbf{u}_{opt}(\mathbf{d})$  gives

$$J(\mathbf{u}, \mathbf{d}) = \underbrace{J(\mathbf{u}_{opt}(\mathbf{d}), \mathbf{d})}_{\mathbf{J}_{opt}(\mathbf{d})} + \underbrace{\mathbf{J}_u}_{=0}(\mathbf{u} - \mathbf{u}_{opt}) + \frac{1}{2}(\mathbf{u} - \mathbf{u}_{opt})^T \mathbf{J}_{uu}(\mathbf{u} - \mathbf{u}_{opt}) \quad (2.7)$$

Here  $\mathbf{J}_u = 0$  (since we are expanding around an optimal point), so we get the following expression for the loss

$$L(u, d) = J(u, d) - J_{opt}(d) = \frac{1}{2} \mathbf{z}^T \mathbf{z} = \frac{1}{2} \|\mathbf{z}\|_2^2 \quad (2.8)$$

where

$$\mathbf{z} = \mathbf{J}_{uu}^{1/2}(\mathbf{u} - \mathbf{u}_{opt}(d)) \quad (2.9)$$



### 2.3.3 Expression for $J_u$ around Moving $\mathbf{u}_{opt}(d)$

A similar expansion, but now of the gradient gives

$$\mathbf{J}_u = \underbrace{\mathbf{J}_u(\mathbf{u}_{opt})}_{=0} + \mathbf{J}_{uu}(\mathbf{u} - \mathbf{u}_{opt}) = \mathbf{J}_{uu}(\mathbf{u} - \mathbf{u}_{opt}) \quad (2.10)$$

Combining this with (2.9) gives  $\mathbf{z} = \mathbf{J}_{uu}^{-1/2} \mathbf{J}_u$  and we have the ‘‘Johannes expression’’ for the loss

$$L(u, d) = J(u, d) - J_{opt}(d) = \frac{1}{2} \left\| \mathbf{J}_{uu}^{-1/2} \mathbf{J}_u \right\|_2^2 \quad (2.11)$$

From these expressions we see that minimizing the loss  $L$  is equivalent to minimizing  $\|\mathbf{z}\|_2$ , which is equivalent to minimizing  $\left\| \mathbf{J}_{uu}^{1/2}(\mathbf{u} - \mathbf{u}_{opt}) \right\|_2$  or  $\left\| \mathbf{J}_{uu}^{-\frac{1}{2}} \mathbf{J}_u \right\|_2$ .

Thus, we have the important conclusion that minimizing the loss is equivalent to minimizing the weighted 2-norm of the gradient  $\mathbf{J}_u$ , with the weight being given by the matrix  $\mathbf{J}_{uu}^{-1/2}$ . However, for the ‘‘normal’’ case when there are no restrictions (like fixing some elements to zero) on the matrix  $\mathbf{H}$ , we will show below that the weight  $\mathbf{J}_{uu}^{-1/2}$  does not have any effect on the optimal  $\mathbf{H}$ .

### 2.3.4 Optimal Sensitivities

Note from (2.6) that we can write  $\mathbf{u}_{opt} = \mathbf{F}_u \mathbf{d}$  where  $\mathbf{F}_u = -\mathbf{J}_{uu}^{-1} \mathbf{J}_{ud}$ . More generally, we can write

$$\mathbf{y}^{opt} = \mathbf{F} \mathbf{d} \quad (2.12)$$

where  $\mathbf{F}$  is the optimal sensitivity of the outputs (measurements) with respect to the disturbances. Here,  $\mathbf{F}$  can be obtained using (2.2) and (2.6),

$$\mathbf{y}^{opt} = \mathbf{G}^y \mathbf{u}_{opt} + \mathbf{G}_d^y \mathbf{d} = (-\mathbf{G}^y \mathbf{J}_{uu}^{-1} \mathbf{J}_{ud} + \mathbf{G}_d^y) \mathbf{d}$$

that is,

$$\mathbf{F} = (-\mathbf{G}^y \mathbf{J}_{uu}^{-1} \mathbf{J}_{ud} + \mathbf{G}_d^y) \quad (2.13)$$

However,  $\mathbf{J}_{uu}$  can be difficult to obtain, especially if one relies on numerical methods, and also taking the difference can introduce numerical inaccuracy. Thus, for practical use it is often better to obtain  $\mathbf{F}$  from its definition,  $\mathbf{F} = d\mathbf{y}^{opt}/d\mathbf{d}$ , by numerically reoptimizing the model for the disturbances.

## 2.4 The Loss Method

Now we are finally ready to derive the main results.

### 2.4.1 The Loss Variable $z$ as a Function of Disturbances and Noise

We start from the loss expression in (2.8) with  $\|z\|_2^2$  where  $z = \mathbf{J}_{uu}^{1/2}(\mathbf{u} - \mathbf{u}_{opt})$ . We want to write  $z$  as a function of  $\mathbf{d}$  and  $\mathbf{n}^y$ . The first step is to write  $\mathbf{u} - \mathbf{u}_{opt}$  as a function of  $\mathbf{c} - \mathbf{c}_{opt}$ . We have  $\mathbf{c} = \mathbf{H}\mathbf{y}$ , so

$$\begin{aligned}\mathbf{c} &= \mathbf{H}\mathbf{y} = \mathbf{H}\mathbf{G}^y\mathbf{u} + \mathbf{H}\mathbf{G}_d^y\mathbf{d} \\ \mathbf{c}_{opt} &= \mathbf{H}\mathbf{y}^{opt} = \mathbf{H}\mathbf{G}^y\mathbf{u}_{opt} + \mathbf{H}\mathbf{G}_d^y\mathbf{d}\end{aligned}$$

Thus,  $\mathbf{c} - \mathbf{c}_{opt} = \mathbf{H}\mathbf{G}^y(\mathbf{u} - \mathbf{u}_{opt})$ , or

$$(\mathbf{u} - \mathbf{u}_{opt}) = (\mathbf{H}\mathbf{G}^y)^{-1}(\mathbf{c} - \mathbf{c}_{opt})$$

where  $\mathbf{G} = \mathbf{H}\mathbf{G}^y$  is the transfer function from  $\mathbf{u}$  to  $\mathbf{c}$ .

The next step is to express  $(\mathbf{c} - \mathbf{c}_{opt})$  as a function of  $\mathbf{d}$  and  $\mathbf{n}^y$ . From (2.4) we have that  $\mathbf{H}(\mathbf{y} + \mathbf{n}^y) = \mathbf{c}_s$  (constant), or

$$\mathbf{c} = \mathbf{H}\mathbf{y} = -\mathbf{H}\mathbf{n}^y + \mathbf{c}_s$$

Here,  $\mathbf{c}_s = 0$ , since we assume the nominal point is optimal. From (2.12) we have that  $\mathbf{c}_{opt} = \mathbf{H}\mathbf{F}\mathbf{d}$ . Since the signs for  $\mathbf{n}^y$  and  $\mathbf{d}$  do not matter for the expressions we derive below (we can have both positive and negative changes), we derive

$$\mathbf{c} - \mathbf{c}_{opt} = \mathbf{H}(\mathbf{F}\mathbf{d} + \mathbf{n}^y) = \mathbf{H}(\mathbf{F}\mathbf{W}_d\mathbf{d}' + \mathbf{W}_{n^y}\mathbf{n}^{y'}) = \mathbf{H}[\mathbf{F}\mathbf{W}_d \quad \mathbf{W}_{n^y}] \begin{bmatrix} \mathbf{d}' \\ \mathbf{n}^{y'} \end{bmatrix}$$

Note that  $\mathbf{W}_d$  and  $\mathbf{W}_{n^y}$  are usually diagonal matrices, representing the magnitude of the disturbances and measurement noises, respectively.

### 2.4.2 Loss for Given $H$ , Disturbance and Noise (Analysis)

In summary, we have derived that for the given normalized disturbances  $\mathbf{d}'$  and for the given normalized measurement noises  $\mathbf{n}^{y'}$  the loss is given by

$$L = \frac{1}{2} \mathbf{z}^T \mathbf{z} \quad (2.14)$$

where

$$\mathbf{z} = \mathbf{J}_{uu}^{1/2}(\mathbf{u} - \mathbf{u}_{opt}) = \underbrace{\mathbf{J}_{uu}^{1/2}(\mathbf{H}\mathbf{G}^y)^{-1}\mathbf{H}\mathbf{Y}}_{\mathbf{M}(\mathbf{H})} \begin{bmatrix} \mathbf{d}' \\ \mathbf{n}^{y'} \end{bmatrix} \quad (2.15)$$

$$\mathbf{Y} = [\mathbf{F}\mathbf{W}_d \quad \mathbf{W}_{n^y}] \quad (2.16)$$

### 2.4.3 Worst-case and Average Loss for Given $\mathbf{H}$ (Analysis)

The above expressions give the loss for the given  $\mathbf{d}$  and  $\mathbf{n}^{y'}$ , but the issue is the find the “magnitude” of the loss for the set bounded as

$$\left\| \begin{bmatrix} \mathbf{d}' \\ \mathbf{n}^{y'} \end{bmatrix} \right\|_2 \leq 1 \quad (2.17)$$

Here “magnitude” can be defined in different ways, and the worst case loss (Halvorsen et al, 2003) and average loss (Kariwala et al, 2008) for a given  $\mathbf{H}$  are given by

$$L_{wc} = \frac{1}{2} \bar{\sigma}(\mathbf{M})^2 \quad (2.18)$$

$$L_{avg} = \frac{1}{6(n_y + n_d)} \|\mathbf{M}\|_F^2 \quad (2.19)$$

where

$$\mathbf{M}(\mathbf{H}) = \mathbf{J}_{uu}^{1/2}(\mathbf{H}\mathbf{G}^y)^{-1}\mathbf{H}\mathbf{Y} \quad (2.20)$$

Here  $\bar{\sigma}(\mathbf{M})$  denotes the singular value (induced 2-norm) of the matrix  $\mathbf{M}(\mathbf{H})$ , and  $\|\mathbf{M}\|_F$  denotes the Frobenius norm (normal 2-norm) of the matrix  $\mathbf{M}$ . Use of the norm of  $\mathbf{M}$  to analyze the loss is known as the “exact local method”.

### 2.4.4 Loss Method for Finding Optimal $\mathbf{H}$

The optimal  $\mathbf{H}$  can be found by minimizing either the worst-case loss (2.18) or the average loss (2.19). Fortunately, (Kariwala et al, 2008) prove that the  $\mathbf{H}$  that minimizes the average loss in equation (2.19) is super optimal, in the sense that the same  $\mathbf{H}$  minimizes the worst case loss in (2.18). Hence, only minimization of the Frobenius norm in (2.19) is considered in the rest of the paper. The scaling factor  $\frac{1}{6(n_y+n_d)}$  does not have any effect on the solution

of the equation (2.19) and hence it is omitted in the problem formulation. Similarly, the square does not effect the optimal solution and can be omitted.

In summary, the problem is to find the combination matrix  $\mathbf{H}$  that minimizes the Frobenius norm of  $\|\mathbf{M}\|$  in (2.19), that is,

$$\min_{\mathbf{H}} \left\| \mathbf{J}_{uu}^{1/2} (\mathbf{H}\mathbf{G}^y)^{-1} \mathbf{H}\mathbf{Y} \right\|_F \quad (2.21)$$

where  $\mathbf{Y} = [\mathbf{F}\mathbf{W}_d \ \mathbf{W}_{n^y}]$ . We call this the minimum loss method for finding optimal linear measurement combinations,  $\mathbf{c} = \mathbf{H}\mathbf{y}$ .

The objective in (2.21) is to find the nonsquare  $n_u \times n_y$  matrix  $\mathbf{H}$  (note that  $n_u = n_c$ ). In most cases it may be recast as a convex optimization problem as given in (2.23) below. The exception is if  $\mathbf{H}$  has a specified structure, for example,  $\mathbf{H}$  is a selection matrix, which is discussed in Section 2.6.

### Further Comments

1. Using the norm of  $\mathbf{M}$  to analyze the loss is known as the “exact local method” and finding the optimal  $\mathbf{H}$  is the “exact local method optimization problem”. However, in this paper we simply call it the “loss method”.
2. To include changes in the weights in the cost function  $\mathbf{p}$  (prices), we need to find the optimal sensitivity to price changes,  $\mathbf{y}^{opt} = \mathbf{F}_p \mathbf{p}$ . The corrected setpoint for the variables  $\mathbf{c} = \mathbf{H}\mathbf{y}$  is then

$$\mathbf{c}_s = \mathbf{H}\mathbf{y}^{opt} = \mathbf{H}\mathbf{F}_p \mathbf{p} \quad (2.22)$$

3. The effect (transfer function) from  $\mathbf{c}_s$  to  $\mathbf{z}$  is  $\mathbf{M}_n = \mathbf{J}_{uu}^{1/2} (\mathbf{H}\mathbf{G}^y)^{-1}$ , and from  $\mathbf{c}_s$  to  $\mathbf{u}$  is  $\mathbf{G}^{-1} = (\mathbf{H}\mathbf{G}^y)^{-1}$ . Since there are extra degrees of freedom in  $\mathbf{H}$  which are not set by the optimization problem, either of these ( $\mathbf{M}_n$  or  $\mathbf{G}$ ) can be selected freely; see below for details.

**Exercise 2.1.** Consider a scalar case ( $n_u = n_c = 1$ ) with no disturbances ( $\mathbf{F} = 0$ ) and assume that the measurements  $\mathbf{y}$  have been scaled such that  $\mathbf{W}_{n^y} = I$  (noise of equal magnitude on all outputs). For the scalar case,  $\mathbf{J}_{uu}^{1/2}$  does not matter for the optimization problem which becomes  $\min_{\mathbf{H}} \|(\mathbf{H}\mathbf{G}^y)^{-1} \mathbf{H}\|_F^2$  and we want to find the optimal  $\mathbf{H}$ .

- (a) Consider the case with 2 measurements, so  $\mathbf{G}^y = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}$  (column vector) and  $\mathbf{H} = [h_1 \ h_2]$  (row vector), and solve the problem analytically. Also compute the optimal norm,  $j = \|(\mathbf{H}\mathbf{G}^y)^{-1} \mathbf{H}\|_F$ .
- (b) Derive the result more generally for the scalar case with any number of measurements, by making use of the definition of the induced 2-norm (singular value), which for a vector gives,  $\|\mathbf{G}^y\|_2 = \sqrt{\mathbf{G}^{yT} \mathbf{G}^y} = \max_{\mathbf{h}} \frac{\|\mathbf{G}^y \mathbf{h}\|_2}{\|\mathbf{h}\|_2}$

(note that for a vector the Frobenius norm ( $F$ ) and 2-norm ( $2$ ) are the same).

- (c) Use the analytical formula presented below, to derive the general result for the multivariable case ( $n_u = n_c > 1$ ).

**Solution 2.1.**

- (a) Two measurements.

$$\mathbf{H}\mathbf{G}^y = [h_1 h_2] \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} = h_1 g_1 + h_2 g_2 \quad (\text{scalar})$$

$$j^2 = \|(\mathbf{H}\mathbf{G}^y)^{-1}\mathbf{H}\|_F^2 = (h_1^2 + h_2^2)/(h_1 g_1 + h_2 g_2)^2 = (1 + x^2)/(g_1 + x g_2)^2$$

where  $x = h_2/h_1$ . Set  $d(j^2)/dx = 0$  to find the optimal  $x$ . After a little simplification  $x - (1 + x^2)g_2/(g_1 + x g_2) = 0$  which gives  $x = g_2/g_1$ . Conclusion:

$$h_2/h_1 = g_2/g_1$$

that is, we prefer to control the measurements corresponding to large elements in  $\mathbf{G}^y$ . Also find:  $j^{opt} = 1/\sqrt{(g_1^2 + g_2^2)}$  which we note is equal to  $1/\|\mathbf{G}^y\|_F$

- (b) Any number of measurements: Let  $\mathbf{h} = \mathbf{H}^T$  be a column vector.  $\mathbf{G}^y$  is already a column vector. Since  $\mathbf{H}\mathbf{G}^y = \mathbf{h}^T \mathbf{G}^y$  is a scalar, it is equal to its transpose and we have that  $\mathbf{H}\mathbf{G}^y = \mathbf{G}^{yT} \mathbf{h}$ . Our optimization problem then becomes

$$\min_{\mathbf{H}} j = \min_{\mathbf{h}} \left\| \frac{\mathbf{h}}{(\mathbf{G}^{yT} \mathbf{h})} \right\|_F = 1 / \left( \max_{\mathbf{h}} \frac{\|\mathbf{G}^{yT} \mathbf{h}\|_2}{\|\mathbf{h}\|_2} \right) = 1 / \|\mathbf{G}^y\|_2$$

We have here made use of the induced 2-norm and the fact that both the Frobenius- and 2-norm are the same for a vector. Thus the optimal  $j$  is the inverse of the 2-norm of  $\mathbf{G}^y$ , which generalizes the solution found for the case with two measurements. The optimal  $\mathbf{h} = c\mathbf{G}^y$  (where  $c$  is any scalar since only the relative magnitudes matter), that is,

$$\mathbf{H}^T = c\mathbf{G}^y$$

which generalizes the result above.

- (c) Multivariable case ( $c$  is no longer required to be a scalar). From (2.25) we derive with  $\mathbf{Y} = \mathbf{I}$  ( $\mathbf{F} = 0$  and measurement noise of magnitude 1 for all outputs) that an optimal solution is

$$\mathbf{H}^T = \mathbf{G}^y$$

which generalizes the results above. Thus, for the case where only measurement noise is a concern, and all the measurements have the same noise

magnitude, the optimal is to select the measurements according the gain matrix  $\mathbf{G}^y$ . This means that “sensitive” measurements, with large elements in  $\mathbf{G}^y$  are preferred for control.

## 2.5 Reformulation of Loss Method to Convex Problem and Explicit Solution

We consider here the “normal” case where  $\mathbf{H}$  is a “full” matrix (with no structural constraints).

**Theorem 2.1 (Reformulation as a convex problem).** *The problem in equation (2.21) may seem non-convex, but for the normal case where  $\mathbf{H}$  is a “full” matrix (with no structural constraints), it can be reformulated as a constrained quadratic programming problem (Alstad et al, 2009)*

$$\begin{aligned} \min_{\mathbf{H}} \|\mathbf{H}\mathbf{Y}\|_F \\ \text{s.t. } \mathbf{H}\mathbf{G}^y = \mathbf{J}_{uu}^{1/2} \end{aligned} \quad (2.23)$$

*Proof.* From the original problem in equation (2.21), we have that the optimal solution  $\mathbf{H}$  is non-unique because if  $\mathbf{H}$  is a solution then  $\mathbf{H}_1 = \mathbf{D}\mathbf{H}$  is also a solution for any non-singular matrix  $\mathbf{D}$  of size  $n_c \times n_c$ . This follows because

$$\mathbf{J}_{uu}^{1/2}(\mathbf{H}_1\mathbf{G}^y)^{-1}\mathbf{H}_1\mathbf{Y} = \mathbf{J}_{uu}^{1/2}(\mathbf{H}\mathbf{G}^y)^{-1}\mathbf{D}^{-1}\mathbf{D}\mathbf{H}\mathbf{Y} = \mathbf{J}_{uu}^{1/2}(\mathbf{H}\mathbf{G}^y)^{-1}\mathbf{H}\mathbf{Y}$$

One implication is that we can freely choose  $\mathbf{G} = \mathbf{H}\mathbf{G}^y$ , which is a  $n_c \times n_c$  matrix representing the effect of  $\mathbf{u}$  on  $\mathbf{c}$  ( $\mathbf{c} = \mathbf{G}\mathbf{u}$ ). Thus, in (2.21) we may use the non-uniqueness of  $\mathbf{H}$  to set the first part of the expression equal to the identity matrix, which is equivalent to setting  $\mathbf{H}\mathbf{G}^y = \mathbf{J}_{uu}^{1/2}$ . This identity must then be added as a constraint in the optimization as shown in (2.23).  $\square$

The reason for the non-uniqueness is that since  $n_y \geq n_c$ ,  $\mathbf{H}$  is “fat”  $n_c \times n_y$  matrix (with more columns than rows).

**Theorem 2.2 (Analytical solution).** *Under the assumption that  $\mathbf{Y}\mathbf{Y}^T$  is full rank, an analytical solution for the problem in (2.23) is (Alstad et al, 2009)*

$$\mathbf{H}^T = (\mathbf{Y}\mathbf{Y}^T)^{-1}\mathbf{G}^y(\mathbf{G}^{yT}(\mathbf{Y}\mathbf{Y}^T)^{-1}\mathbf{G}^y)^{-1}\mathbf{J}_{uu}^{1/2} \quad (2.24)$$

*Proof.* The result is proved in (Alstad et al, 2009) and is based on first vectorizing the problem and then using standard results from constrained quadratic optimization.

The analytical solution in Theorem 2.2, results in a  $\mathbf{H}$  satisfying  $\mathbf{H}\mathbf{G}^y = \mathbf{J}_{uu}^{1/2}$ . However, recall that the optimal solution  $\mathbf{H}$  is non-unique, and we may use it to derive a simplified analytical solution.

**Theorem 2.3 (Simplified analytical solution).** *Under the assumption that  $\mathbf{Y}\mathbf{Y}^T$  is full rank, another analytical solution for the problem in (2.23) is*

$$\mathbf{H}^T = (\mathbf{Y}\mathbf{Y}^T)^{-1}\mathbf{G}^y \quad (2.25)$$

*Proof.* This follows trivially from Theorem 2.2, since if  $\mathbf{H}^T$  is a solution then so is  $\mathbf{H}_1^T = \mathbf{H}^T\mathbf{D}$  and we simply select  $\mathbf{D} = (\mathbf{G}^{y^T}(\mathbf{Y}\mathbf{Y}^T)^{-1}\mathbf{G}^y)^{-1}\mathbf{J}_{uu}^{1/2} = \mathbf{J}_{uu}^{-1/2}\mathbf{G}^{y^T}(\mathbf{Y}\mathbf{Y}^T)^{-1}\mathbf{G}^y$ , which is a  $n_c \times n_c$  matrix.  $\square$

Note that the analytical expressions in Theorems 2.2 and 2.3 require  $\mathbf{Y}\mathbf{Y}^T$  to be full rank. This implies that they generally do not apply to the case with no measurement error,  $\mathbf{W}_{ny} = 0$ , but otherwise they apply for any number of measurements. One exception (but not so common in practice), when the analytical expressions for  $\mathbf{H}$  do apply also for  $\mathbf{W}^y = 0$ , is when  $n_y \leq n_d$ , because  $\mathbf{Y}\mathbf{Y}^T$  then remains full rank.

**Corollary 2.1 (Important insight).** *Theorem 2.3 gives the very important insight that  $\mathbf{J}_{uu}$  is not needed for finding the optimal  $\mathbf{H}$ , provided we have the normal case where  $\mathbf{H}$  can be any  $n_c \times n_y$  matrix.*

*This means that in (2.21) we can replace  $\mathbf{J}_{uu}^{1/2}$  by any non-singular matrix, and still get an optimal  $\mathbf{H}$ . This can greatly simplify practical calculations, because  $\mathbf{J}_{uu}$  may be difficult to obtain numerically because it involves the second derivative. On the other hand, we found that  $\mathbf{F}$ , which enters in  $\mathbf{Y}$ , is relatively straightforward to obtain numerically. Although  $\mathbf{J}_{uu}$  is not needed for finding the optimal  $\mathbf{H}$ , it would be required for finding a numerical value for the loss.*

The analytical solutions are useful, in particular for their insights they yield, but for practical calculations it is usually faster and more robust to compute the optimal  $\mathbf{H}$  by solving the convex quadratic optimization problems. In addition, the convex optimization problems do not need the requirement that  $\mathbf{Y}\mathbf{Y}^T$  is non-singular. Based on the insight in Corollary 2.1, the quadratic optimization in Theorem 2.1 (Alstad et al, 2009), can be further reformulated to a more general form (Yelchuru and Skogestad, 2010)

**Theorem 2.4 (Generalized convex formulation).** *An optimal  $\mathbf{H}$  for the problem in (2.23) is*

$$\begin{aligned} \min_{\mathbf{H}} \|\mathbf{H}\mathbf{Y}\|_F \\ \text{s.t. } \mathbf{H}\mathbf{G}^y = \mathbf{Q} \end{aligned} \quad (2.26)$$

where  $\mathbf{Q}$  is any non-singular  $n_c \times n_c$  matrix, for example,  $\mathbf{Q} = \mathbf{I}$ , but  $\mathbf{Q}$  must be fixed while minimizing  $\|\mathbf{H}\mathbf{F}\|_F$ .

*Proof.* The result follows from Corollary 2.1, but can more generally be de-

rived as follows. The problem in (2.23) is to minimize  $\left\| \underbrace{(\mathbf{J}_{uu}^{1/2}(\mathbf{H}\mathbf{G}^y)^{-1}\mathbf{H}\mathbf{Y})}_{\mathbf{X}} \right\|_F$ .

The reason why we can omit the  $n_c \times n_c$  matrix  $\mathbf{X}$ , is that if  $\mathbf{H}$  is an optimal solution then so is  $\mathbf{H}_1 = \mathbf{D}\mathbf{H}$  where  $\mathbf{D}$  is any nonsingular  $n_c \times n_c$  (see proof of Theorem 2.1). However, note that the matrix  $\mathbf{X}$ , or equivalently the matrix  $\mathbf{Q}$ , must be fixed during the optimization, so it needs to be added as a constraint.  $\square$

The fact that  $\mathbf{Q}$  can be chosen freely (Theorem 2.4) can be useful for numerical reasons, or finding improved bounds for cases with constraints on  $\mathbf{H}$  (see below).

Once we have found an optimal  $\mathbf{H}$  using any of the Theorems above, we can use the non-uniqueness of optimal  $\mathbf{H}$  to find another  $\mathbf{H}_1 = \mathbf{D}\mathbf{H}$  with desired property or structure. For example, one can select  $\mathbf{D}$  such that  $\mathbf{G} = \mathbf{H}\mathbf{G}^y = \mathbf{I}$ . Alternatively, one can specify selected elements in  $\mathbf{H}_1$ , for example,  $\mathbf{H}_1 = [\mathbf{I} \quad \mathbf{K}]$ . In the latter case, write  $\mathbf{H} = [\mathbf{H}_l \mathbf{H}_r]$  and assume  $\mathbf{H}_l$  is full rank, then  $\mathbf{H}_1 = [\mathbf{I} \quad \mathbf{K}] = [\mathbf{D}\mathbf{H}_l \quad \mathbf{D}\mathbf{H}_r]$ , and we find  $\mathbf{d}_b = \mathbf{H}_l^{-1}$  and  $\mathbf{K} = \mathbf{H}_l^{-1}\mathbf{H}_r$ .

### Required information

To find the optimal “full”  $\mathbf{H}$  using the loss method we need four pieces of information. First, for the measurements we need the optimal disturbance sensitivity ( $\mathbf{F}$ ) and input sensitivity ( $\mathbf{G}^y$ ). These are obtained from the model. Next, we must specify the disturbance magnitudes ( $W_d$ ) and the noise magnitudes ( $W_{ny}$ ). The matrix  $\mathbf{J}_{uu}$  is not needed except when there are structural constraints, as discussed in the next section.

Note that changes (disturbances) in the prices (parameters) in the cost function do not change the optimal  $\mathbf{H}$ , based on the assumption that we still have a quadratic optimization problem with constant weights. However, as given in (2.22) the setpoint for  $\mathbf{c}$  needs to be adjusted,  $\mathbf{c}_s = \mathbf{H}\mathbf{F}_p\mathbf{p}$  and for this we need for the measurements the optimal price sensitivity ( $\mathbf{F}_p$ ) which can be obtained from the model.

## 2.6 Structural Constraints on $\mathbf{H}$

In the previous section we considered the normal case where  $\mathbf{H}$  may be any “full” matrix. In terms of selecting controlled variables,  $\mathbf{c} = \mathbf{H}\mathbf{y}$ , this means that any combination of measurements are allowed. However, in practice there may be constraints on  $\mathbf{H}$ , for example, one wants to use a subset of the measurements or one want to use a decentralized structure for  $\mathbf{H}$ .

We will consider the following special cases

**Case 1.** No restrictions on  $\mathbf{H}$ . This is the case already considered where Theorems 2.1– 2.4 hold.



Note that key for deriving Theorems 2.1– 2.4 was that if  $\mathbf{H}$  is a solution then so is  $\mathbf{H}_1 = \mathbf{D}\mathbf{H}$  where  $\mathbf{D}$  is any non-singular matrix.

**Case 2.**  $\mathbf{H}$  contains a subset of the measurements but is otherwise full. Theorems 2.1– 2.4 hold also in this case.

The reason is that  $\mathbf{H}_1 = \mathbf{D}\mathbf{H}$  will have the same structure as  $\mathbf{H}$  for any nonsingular  $\mathbf{D}$ . This is because if  $\mathbf{H}$  has columns equal to zero, then these columns will remain zero in  $\mathbf{D}\mathbf{H}$  even if  $\mathbf{D}$  is “full”. For example, if  $\mathbf{H} = \begin{bmatrix} 0 & \mathbf{x} & 0 & \mathbf{x} & \mathbf{x} \\ 0 & \mathbf{x} & 0 & \mathbf{x} & \mathbf{x} \end{bmatrix}$  then we can still allow a full  $\mathbf{D} = \begin{bmatrix} \mathbf{x} & \mathbf{x} \\ \mathbf{x} & \mathbf{x} \end{bmatrix}$  (where  $\mathbf{x}$  is any number) and keep the structure of  $\mathbf{H}$  in  $\mathbf{H}_1 = \mathbf{D}\mathbf{H}$ .

**Case 3.**  $\mathbf{H}$  contains measurements from disjoint set, so  $\mathbf{H}$  has a block diagonal (decentralized) structure. Theorems 2.1– 2.4 do *not* hold in this case.

The reason is that for  $\mathbf{H}_1 = \mathbf{D}\mathbf{H}$  to have the same structure,  $\mathbf{D}$  must have a structure similar to  $\mathbf{H}$ . For example, let  $\mathbf{H} = \begin{bmatrix} \mathbf{x} & \mathbf{x} & \mathbf{x} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{x} & \mathbf{x} \end{bmatrix}$  then  $\mathbf{D} = \begin{bmatrix} \mathbf{x} & 0 \\ 0 & \mathbf{x} \end{bmatrix}$  (where  $\mathbf{x}$  is any number) and if  $\mathbf{H} = \begin{bmatrix} \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & 0 & 0 & \mathbf{x} & \mathbf{x} \end{bmatrix}$  then  $\mathbf{D} = \begin{bmatrix} \mathbf{x} & \mathbf{x} \\ 0 & \mathbf{x} \end{bmatrix}$ .

Thus, for case 3 we do not have a convex problem formulation, that is, we need to solve the nonconvex problem in (2.21) (with additional constraints on the structure of  $\mathbf{H}$ ). This is not surprising as decentralized control is generally a nonconvex problem. Nevertheless, Theorems 2.1 and 2.4, with additional constraints on the structure of  $\mathbf{H}$ , give convex optimization problems that provide upper bounds on the optimal  $\mathbf{H}$  for case 3. In particular, in Theorem 2.4 4 we may make use of the extra degree of freedom provided by the matrix  $\mathbf{Q}$  (Yelchuru and Skogestad, 2010).

Also note that, as opposed to cases 1 and 2,  $J_{uu}$  is needed to find the optimal solution for case 3. This may seem a bit surprising.

**Case 4.** Decentralized control using single measurements, that is  $n_{ys} = n_c$  where  $n_{ys}$  is the number of selected measurements). Theorems 2.1– 2.4 hold also in this case.

This is a special case of case 3 where we use the fewest number of measurements. This case is different from case 2 in that  $\mathbf{H}$  is a diagonal matrix. The reason why Theorems 2.1– 2.4 hold in this case, is that we can still keep  $\mathbf{D}$  full because the “non-zero” part of  $\mathbf{H}$  is square and we can to change it to anything, so we can treat it a special case of “full  $\mathbf{H}$ ”.

## 2.7 Some Special Cases: Nullspace Method and Maximum Gain Rule

The general optimization problem is

$$\min_{\mathbf{H}} \left\| \mathbf{J}_{uu}^{1/2} (\mathbf{H}\mathbf{G}^y)^{-1} \mathbf{H}\mathbf{Y} \right\|_F \quad (2.27)$$

where  $\mathbf{Y} = [\mathbf{F}\mathbf{W}_d\mathbf{W}_{ny}]$ . The objective is to find the nonsquare  $n_c \times n_y$  matrix  $\mathbf{H}$  (note that  $n_u = n_c$ ). We will here consider some special cases of this problem, which historically were developed before the convex and analytical solutions presented above.

### 2.7.1 No Measurement Noise: Nullspace Method (“full $\mathbf{H}$ ”)

For the special case with no measurement noise,  $\mathbf{W}_{ny} = 0$ , and with more (independent) measurements than (independent) inputs and disturbances,  $n_y \geq n_u + n_d$ , it is possible to find  $\mathbf{H}$  such that

$$\mathbf{H}\mathbf{F} = 0 \quad (2.28)$$

that is, the loss is zero. This is called the “nullspace method” (Alstad and Skogestad, 2007) because  $\mathbf{H}$  is in the nullspace of  $\mathbf{F}$ . In this case,  $\mathbf{G}_y$  and  $\mathbf{W}_d$  do not matter for finding the optimal  $\mathbf{H}$ .

The nullspace method is very simple and has been found to be very useful in applications. Since the nullspace method neglects the effect of measurement error, it is important to use preselect a subset of the measurements that are expected to be insensitive to measurement errors.

Also, one cannot include too many disturbances, because otherwise one cannot satisfy the requirement  $n_y \geq n_u + n_d$ .

One limitation with the analytical formulas in (2.24) and (2.25) is that they do not give the nullspace method as a special case. This is because  $\mathbf{Y} = [\mathbf{F}\mathbf{W}_d \ \mathbf{W}_{ny}]$  at most has rank  $n_d$  when  $\mathbf{W}_{ny} = 0$ . Thus, the  $n_y \times n_y$  matrix  $\mathbf{Y}\mathbf{Y}^T$  at most has rank  $n_d$  and is not invertible because this would require the rank to be  $n_y$ . However, the convex optimization problems in Theorems 2.1 and 2.4 do give the nullspace method as a special case.

Comment: In general, with measurement noise included or with few measurements (so  $n_y < n_u + n_d$ ), it is not possible to make  $\mathbf{H}\mathbf{Y}$  zero.

#### Explicit Expression for $\mathbf{H}$ for Nullspace Method

The following explicit expression applies for  $\mathbf{H}$  (Alstad and Skogestad, 2007):

$$\mathbf{H} = [\mathbf{J}_{uu}\mathbf{J}_{ud}](\tilde{\mathbf{G}}^y)^{-1} \quad (2.29)$$

*Proof.* Here is a proof which is much simpler than that given in (Alstad and Skogestad, 2007): Want to find  $\mathbf{c} = \mathbf{H}\mathbf{y}$  with zero loss.

1. Measurement relationship:  $\mathbf{y} = \tilde{\mathbf{G}}^y \begin{bmatrix} \mathbf{u} \\ \mathbf{d} \end{bmatrix}$ . Inverting this:

$$\begin{bmatrix} \mathbf{u} \\ \mathbf{d} \end{bmatrix} = (\tilde{\mathbf{G}}^y)^{-1} \mathbf{y} \quad (2.30)$$

2. Optimality condition (NCO):

$$\mathbf{J}_u = 0 \quad (2.31)$$

3. First-order expansion of gradient:

$$\mathbf{J}_u = \mathbf{J}_u^* + \mathbf{J}_{uu}^* \mathbf{u} + \mathbf{J}_{ud}^* \mathbf{d} = [\mathbf{J}_{uu}^* \quad \mathbf{J}_{ud}^*] \begin{bmatrix} \mathbf{u} \\ \mathbf{d} \end{bmatrix}$$

where we use  $\mathbf{J}_{u^*} = 0$ .

4. (2.30) and (2.31) then give:  $[\mathbf{J}_{uu} \quad \mathbf{J}_{ud}] \tilde{\mathbf{G}}_y^{-1} \mathbf{y} = 0$  and it follows that  $\mathbf{H} = [\mathbf{J}_{uu} \quad \mathbf{J}_{ud}] (\tilde{\mathbf{G}}^y)^{-1}$ .  $\square$

### 2.7.2 No Disturbances

The case with no disturbances has limited practical significance, but is nevertheless and interesting limiting cases.

We assume there are no disturbances,  $\mathbf{W}_d = 0$ , and we scale the measurements  $\mathbf{y}$  so that they all have unity measurement noise,  $\mathbf{W}_{n^y} = I$ . From the analytical expression (2.25), we then have that an optimal solution is

$$\mathbf{H}^T = \mathbf{G}^y \quad (2.32)$$

This gives the important insight that we prefer sensitive measurements.

### 2.7.3 An Approximate Analysis Method for the General Case: “Maximum Gain Rule”

The maximum gain rule is an approximate analysis method for a given  $\mathbf{H}$ . If we want to compare (analyze) alternative choices for  $\mathbf{H}$ , for example, alternative choices of individual measurements, then the “maximum gain rule” is effective and gives considerable insight. The maximum gain rule has also been used to find “optimal”  $\mathbf{H}$ , especially for the case where one wants to control individual measurements, and Yu and Kariwala have devised efficient branch and bound methods for solving this problem.

In the “maximum gain rule” one considers the scaled gain matrix  $\mathbf{G} = \mathbf{H}\mathbf{G}^y$  from  $\mathbf{u}$  to  $\mathbf{c}$ . To derive the maximum gain rule, we return to the loss expression

$$J = \frac{1}{2} \mathbf{z}^T \mathbf{z}$$

where

$$\begin{aligned} \mathbf{z} &= \mathbf{J}_{uu}^{1/2}(\mathbf{u} - \mathbf{u}_{opt}) = \mathbf{J}_{uu}^{1/2} \mathbf{G}^{-1}(\mathbf{c} - \mathbf{c}_{opt}) \\ \mathbf{c} - \mathbf{c}_{opt} &= \mathbf{H}(\mathbf{F}\mathbf{d} + \mathbf{n}^y) \\ \mathbf{G} &= \mathbf{H}\mathbf{G}^y \end{aligned}$$

Here,  $\mathbf{c} - \mathbf{c}_{opt}$  may be viewed as the “optimal” (or expected) variation in the selected variables,  $\mathbf{c} = \mathbf{H}\mathbf{y}$ , caused by disturbances and measurement noise. The magnitude of  $\mathbf{c} - \mathbf{c}_{opt} = \mathbf{H}\mathbf{F}\mathbf{d} + \mathbf{H}\mathbf{n}^y$  is obtained by adding the magnitude of the contributions from  $\mathbf{H}\mathbf{F}\mathbf{d}$  and  $\mathbf{H}\mathbf{n}^y$ , and we assume in the following that  $\mathbf{c} - \mathbf{c}_{opt} = \mathbf{W}_c \mathbf{c}'$  where  $\mathbf{W}_c$  is a diagonal matrix for the expected optimal variation (“optimal span”) in  $\mathbf{c}$  and we assume that all  $\|\mathbf{c}'\|_2 \leq 1$  are allowed.  $\mathbf{c} - \mathbf{c}_{opt}$  translates into changes in the inputs ( $\mathbf{u} - \mathbf{u}_{opt}$ ) by the transformation  $\mathbf{u} = \mathbf{G}^{-1} \mathbf{c}$  and to a loss through the matrix  $\mathbf{J}_{uu}^{1/2}$ . We want  $(\mathbf{u} - \mathbf{u}_{opt})$  small, so we want the norm of  $\mathbf{G}^{-1}$  small. More specifically, the largest (worst-case) value of  $\|\mathbf{z}\|_2$  for any allowed  $\|\mathbf{c}'\|_2 \leq 1$  is equal to  $\bar{\sigma}(\mathbf{J}_{uu}^{1/2} \mathbf{G}^{-1} \mathbf{W}_c)$ , and we want this as small as possible. From singular value properties we have that the  $\bar{\sigma}(\mathbf{A}^{-1}) = 1/\underline{\sigma}(\mathbf{A})$ , that is we want to maximize  $\text{smin}(\mathbf{W}_c^{-1} \mathbf{G} \mathbf{J}_{uu}^{-1/2})$ .

We have then derived the **maximum gain rule**: *Under the assumption that  $\|\mathbf{c}'\|_2 \leq 1$ , the worst-case loss is given by  $L_{max} = \frac{1}{2} \frac{1}{\underline{\sigma}^2(\mathbf{G}_s)}$  where*

$$\mathbf{G}_s = \mathbf{S}_1 \mathbf{G} \mathbf{S}_2 \quad (2.33)$$

and

$$\begin{aligned} \mathbf{S}_1 &= \mathbf{W}_c^{-1} = \text{diag}(1/|\mathbf{c}_i - \mathbf{c}_{opt,i}|) \\ \mathbf{S}_2 &= \mathbf{J}_{uu}^{-1/2} \end{aligned}$$

Note that  $\mathbf{S}_1$  includes the sum of the optimal variation (as given by the  $\mathbf{F}$ -matrix) and the expected measurement error. Thus, to minimize the loss we should select  $\mathbf{c} = \mathbf{G}\mathbf{u}$  with a large minimum singular value of the scaled gain matrix  $\mathbf{G}_s$ .

The only “non-exact” step in deriving this rule comes from the assumption that all  $\|\mathbf{c}'\|_2 \leq 1$  are allowed, which means that we neglect some of the variations in  $(\mathbf{c} - \mathbf{c}_{opt})$  that are correlated. Nevertheless, since the presence of measurement noise means that there is always some uncorrelated variation, at least if we consider individual measurements,  $\mathbf{c} = \mathbf{y}$ , this implies that we can safely exclude candidate  $\mathbf{c}$ 's with a small gain, that is, with a small value of  $\underline{\sigma}(\mathbf{G}_s)$ .

Note that  $\mathbf{J}_{uu}$  enters into the maximum gain rule, whereas it is actually not required when  $\mathbf{H}$  is the optimal full matrix, see (2.26). However, in general  $\mathbf{J}_{uu}$  must be included, see case 3 in the discussion following (2.26).

- Do we need the maximum gain rule?

One can analyze the loss with alternatives choices for  $\mathbf{c}$  using the “exact local method”, so why do we need the maximum gain rule? The motivation for using the maximum gain rule is at least threefold

1. It is simpler to compute.
2. It given insight, in particular that we want to control “sensitive” variables with a large scaled gain: *Select variables  $\mathbf{c}$  where the “optimal variation” ( $\mathbf{c} - \mathbf{c}_{opt}$ ) (from disturbances, and including measurement noise) is small compared to the “achievable variation” ( $\mathbf{c} = \mathbf{G}\mathbf{u}$ ) (from inputs).*
3. The scaled gain matrix is  $\mathbf{G}_s = S_1\mathbf{G}S_2$ . Here, the gain matrix  $\mathbf{G}$  is obtained by linearizing in a single operating point. To find the “scaling” matrices  $S_1$  and  $S_2$  we need to reoptimize for the disturbances (to find  $\mathbf{c} - \mathbf{c}_{opt}$  needed for  $S_1$ ) and to find the second derivative with respect to the inputs (to find  $S_2 = \mathbf{J}_{uu}^{-1/2}$ ), which can be rather involved calculations. If this information missing, then one may often get good results by estimating the optimal variations to find  $S_1$  (for example, based on operating data) and by setting  $S_2 = I$  (one should in this case scale the inputs so that their expected effect on the cost is similar).

This maximum gain rule has also proven to work surprisingly well on many applications. Nevertheless, if one has data for the optimal sensitivity ( $\mathbf{F}$ ), then our recommendation is to use the “exact local method” instead of the maximum gain rule. This is because one can analyze alternative choices for  $\mathbf{c}$  (and  $\mathbf{H}$ ) more exactly by computing the norm (Frobenius norm or max. singular value) of  $\mathbf{M} = \left[ \mathbf{J}_{uu}^{1/2}(\mathbf{H}\mathbf{G}^y)^{-1}\mathbf{H}\mathbf{Y} \right]$ . Kariwala and Cao (2009) have derived efficient branch and bound algorithms for finding the measurement choice (optimal structured  $\mathbf{H}$ ) that minimize either the norm of  $\mathbf{M}$  as well as the scaled gain,  $\underline{\sigma}(\mathbf{G}_s)$ . Although the computation times for minimizing the latter are somewhat lower, the benefit is not sufficiently large to justify using the maximum gain rule, provided we have the necessary information available for the first method.

## 2.8 Indirect Control and Estimation of Primary Variable

These two problems are very similar, and can be written as a special case of the loss method, involving the same matrices.

### 2.8.1 Indirect Control of $\mathbf{y}_1$

The objective is to keep the primary output  $\mathbf{y}_1$  close to its setpoint, so the cost function is

$$J = (\mathbf{y}_1 - \mathbf{y}_{1,s})^2$$

However,  $\mathbf{y}_1$  is not measured, but we have to use some other measurements  $\mathbf{y}$ . Thus, we want to achieve indirect control of  $\mathbf{y}_1$  by keeping  $\mathbf{c} = \mathbf{H}\mathbf{y}$  at a given setpoint.

To find the optimal “full”  $\mathbf{H}$  using the loss method we need four pieces of information;  $\mathbf{F}, \mathbf{G}^y, \mathbf{W}_d, \mathbf{W}_{ny}$ . In our case, the optimal sensitivity is  $\mathbf{F} = (d\mathbf{y}^{opt}/dd) = (d\mathbf{y}/dd)_{\mathbf{y}_1}$ . It may be obtained by simulations where we keep  $\mathbf{y}_1$  constant for the various disturbances. Instead of using simulations, we may write  $\mathbf{y}_1 = \mathbf{G}_1\mathbf{u} + \mathbf{G}_{d1}\mathbf{d}$ , and then (Hori et al, 2005)  $\mathbf{J}_{uu} = \mathbf{G}_1^t\mathbf{G}_1$ ,  $\mathbf{J}_{ud} = \mathbf{G}_1^t\mathbf{G}_{d1}$ ,  $\mathbf{F} = (-\mathbf{G}^y\mathbf{J}_{uu}^{-1}\mathbf{J}_{ud} + \mathbf{G}_d^y) = (-\mathbf{G}^y\mathbf{G}_1^{-1}\mathbf{G}_{d1} + \mathbf{G}_d^y)$ .

In addition, we need to handle setpoint changes for the primary variable,  $\mathbf{y}_{1,s}$ , which requires changes in the setpoint for  $\mathbf{c}$ . Note that  $\mathbf{y}_{1,s}$  only affects the cost function and may be viewed as a price variable  $p$ , so from (2.22) the required change in the setpoint is  $\Delta\mathbf{c}_s = \mathbf{H}\mathbf{F}_p\Delta\mathbf{y}_{1,s}$ , where  $\mathbf{F}_p$  may be obtained from the model (exercise: derive the expression!).

### 2.8.2 Indirect Control of $\mathbf{y}_1$ Based on Estimator

Note that we still have not used the available degrees of freedom in  $\mathbf{H}$ . To simplify the setpoint adjustment, we may use the degrees of freedom in  $\mathbf{H}$  to make  $\mathbf{H}\mathbf{F}_p = \mathbf{I}$ , or equivalently,  $\mathbf{c} = \mathbf{y}_1$ . This means that  $\mathbf{c}$  should be an estimate of  $\mathbf{y}_1$ . Note that  $\mathbf{y}_1 = \mathbf{G}_1\mathbf{u}$  and  $\mathbf{c} = \mathbf{H}\mathbf{G}_1\mathbf{u}$  (setting  $d = 0$  for simplicity). These two gains need to be identical, so we use the extra degrees of freedom in  $\mathbf{H}$  to make

$$\mathbf{H}\mathbf{G}_1 = \mathbf{G}_1 \quad (2.34)$$

It is then easy to include changes in setpoint; we just control  $\mathbf{c}$  at  $\mathbf{y}_{1,s}$ .

#### Some comments on this estimator

- What kind of estimator is this? If we look at the problem formulation, then we see that it is be the optimal estimator in the following sense: When we control  $\mathbf{y}_1$  at the estimate (using the unconstrained degrees of freedom) then this minimizes the deviation from the given value (setpoint), for the expected range of disturbances and measurement noise.
- For practical purposes, when obtaining the model, it may be smart to let the primary outputs be the degrees of freedom,  $\mathbf{u} = \mathbf{y}_1$  that is, to use

“closed-loop data” (this may seem strange, but it is OK). Then we have  $\mathbf{G}_1 = I$ .

## 2.9 Estimator for $\mathbf{y}_1$ Based on Data

The idea is to use the same approach as for the previous problem, but using data instead of a model. The objective is to estimate  $\mathbf{y}_1$  based on measurements  $\mathbf{y}$  (which we from now on will call  $\mathbf{x}$  to follow statistics notation). That is, we want to find

$$\mathbf{y}_1 = \mathbf{H}\mathbf{x}$$

(where  $\mathbf{x} = \mathbf{y}$ ). The available information is given by the data  $\mathbf{Y}_{all} = [\mathbf{Y}_1; \mathbf{X}]$ . Note that the data must first be centered.

To use our method, we first need to know the expected optimal variation  $Y$ . Here “optimal” means that  $\mathbf{y}_1$  is constant. In addition, we also need to obtain  $\mathbf{G}^y$  and  $\mathbf{G}_1$  from the data. This means that the data must contain “non-optimal” variations in  $\mathbf{u}$ , and not only contain optimal data where  $\mathbf{u} = \mathbf{u}_{opt}(\mathbf{d})$ .

Comment: The setup is the same as for the previous problem, expect that it is not clear how noise in  $\mathbf{y}_1$  can be included. It is a bit similar to “implementation error” which has been neglected since we assumed integral action.

### 2.9.1 Data Approach 1

Here we assume that  $\mathbf{Y}_{all}$  is obtained from two different sources of data.

1. “Optimal” data with constant  $\mathbf{y}_1$  ( $\mathbf{X} = \mathbf{Y}_{opt}$ ): This is closed-loop data for  $\mathbf{y}$  with  $\mathbf{y}_1$  constant for various disturbances ( $\mathbf{d}$ ) and also with noise. It should be representative data for the expected operation. This directly gives the matrix  $\mathbf{Y} = \mathbf{Y}_{opt}$  (including the weights) needed in (2.26).
2. “Non-optimal” data with constant  $\mathbf{d}$ : This is data for  $\mathbf{x}$  and  $\mathbf{y}_1$  collected with varying  $\mathbf{u}$ . From this data we directly obtain  $\mathbf{G}^y$  and  $\mathbf{G}_1$ . By selecting  $\mathbf{u} = \mathbf{y}_1$ , one may also here used closed-loop data (but with  $\mathbf{y}_1$  varying), in this case  $\mathbf{G}_1 = I$ .
3. Find optimal  $\mathbf{H}$  using (2.26) with  $\mathbf{H}\mathbf{G}^y = I$ .

### 2.9.2 Data Approach 2: Loss Regression

More generally, we do not have separate “optimal” and “non-optimal” data. Instead, we have combined data  $\mathbf{Y}_{all}$  where  $\mathbf{y}_1$  and  $\mathbf{x}$  vary simultaneously.

Note that we here use the notation from statistics/chemometrics and call the measurements  $\mathbf{y}$  for  $\mathbf{x}$ .

We can then do a two-step procedure. In the first step, we “split up” the data  $\mathbf{Y}_{all}$  data to find  $\mathbf{G}_1$ ,  $\mathbf{G}^y$  and  $\mathbf{Y}_{opt}$ , and in step 2 we proceed as “normal” to find the optimal  $\mathbf{H}$ .

**Step 1A.** We rearrange the data  $\mathbf{Y}_{all}$  such that the  $\mathbf{y}_1$  values are in the first rows, and the  $\mathbf{x} = \mathbf{y}$ -measurements are in the rest (called  $\mathbf{X}$ ),

$$\mathbf{Y}_{all} = [\mathbf{Y}_1; \mathbf{X}]$$

**Step 1B.** We now want to separate the data into “optimal” and “nonoptimal” data. The data can generally be transformed by multiplying by a (real) unitary matrix  $\mathbf{V}$ , because  $\|[\mathbf{H}\mathbf{Y}_{all}]\| = \|[\mathbf{H}\mathbf{Y}_{all}\mathbf{V}]\|$  for the 2-norm. Thus, we can use the SVD of

$$\mathbf{Y}_1 = \mathbf{U}_1 \mathbf{S}_1 \mathbf{V}_1^T$$

to transform the data to (“split up the data”)

$$\mathbf{Y}_{all}\mathbf{V}_1 = \begin{bmatrix} \mathbf{G}_1 & 0 \\ \mathbf{G}^y & \mathbf{X}_{opt} \end{bmatrix}$$

Now, we have  $\mathbf{G}_1$ ,  $\mathbf{G}^y$  and  $\mathbf{Y} = \mathbf{X}_{opt}$  and can proceed as normal using our derived formulas, see earlier.

**Step 2.** Find the optimal  $\mathbf{H}$  by solving the convex optimization in (2.26) with  $\mathbf{Y} = \mathbf{X}_{opt}$  and the constraint  $\mathbf{H}\mathbf{G}^y = \mathbf{G}_1$ .

```
% Loss method
% step 1A
Yall = [Y1; X];
% step 1B
[u,s,v]=svd(Y1);
Yall1 = Yall*v;
[r1,c1]=size(Yall);
[r2,c2]=size(Y1);
ny=r2;
G1=Yall1(1:ny,1:ny);
Gy=Yall1(ny+1:r1,1:ny);
Xopt = Yall1(ny+1:r1,ny+1:c1);

% step 2,
%Hopt = (pinv(Xopt*Xopt')*Gy)'; %analytical expression
[Hopt,loss]=soc_avg(Gy,Xopt);
D=Hopt*Gy*inv(G1);
Hoptloss=inv(D)*Hopt;
```



## Comments

- Alternatively in step 2, provided  $\mathbf{Y}\mathbf{Y}^T$  has full rank, we may use the analytical expression  $\mathbf{H}^T = (\mathbf{Y}\mathbf{Y}^T)^{-1}\mathbf{G}^y$  in (2.25) and then “rescale”  $\mathbf{H}$  to get a new  $\mathbf{H}_1 = \mathbf{D}\mathbf{H}$  which satisfies  $\mathbf{H}_1\mathbf{G}^y = \mathbf{G}_1$ , which gives  $\mathbf{H}_1 = \mathbf{G}_1(\mathbf{H}\mathbf{G}^y)^{-1}\mathbf{H}$ . If  $\mathbf{Y}\mathbf{Y}^T$  does not have full rank one may use some pseudo inverse (similar to PCR). This adds degrees of freedom to the method. It has not been tested out but some preliminary results are promising.
- The method seems a bit similar to PLS in that we use the data for  $\mathbf{y}_1$  to affect the x-data (we get  $\mathbf{X}_{opt}$  from  $\mathbf{X}$  by using the SVD of  $\mathbf{Y}_1$ , and also use  $\mathbf{G}^y$  when minimizing  $\mathbf{H}\mathbf{X}_{opt}$ ).

### 2.9.2.1 Modification for Case with Too Few Experiments (e.g. Spectroscopic Data)

If we start with a model, then the data matrix  $\mathbf{Y} = [\mathbf{F}\mathbf{W}_d \ \mathbf{W}_{ny}]$  is a “fat” matrix; this is clear since the noise magnitude matrix  $\mathbf{W}_{ny}$  is a square matrix (usually diagonal). Thus, there exists no matrix  $\mathbf{H}$  such that  $\mathbf{H}\mathbf{Y} = 0$ .

However, if we start with data and have many measurements (e.g., spectroscopic data), then  $\mathbf{Y} = \mathbf{X}_{opt}$  is likely a thin matrix, and there will exist an (or actually, infinitely many)  $\mathbf{H}$  such that  $\mathbf{H}\mathbf{Y} = 0$ . Since the experimental data  $\mathbf{Y}$  contains measurement noise, this means that  $\mathbf{H}$  is “fitting” the noise. The proposed method will then be very similar to least squares, although the constraint  $\mathbf{H}\mathbf{G}^y = \mathbf{G}_1$  can make it different (as seen from the numerical examples below).

**Extra step 1C.** To fix this up, one may add “artificial” measurement noise to get a better representation of the actual measurement noise. Since there will always be some independent noise for each measurement, it is suggested to add a diagonal matrix  $\mathbf{W}_{ny}$  to the original data

$$\mathbf{X}_{extended} = [\mathbf{X}_{opt} \ \mathbf{W}_{ny}]$$

where  $\mathbf{X}_{opt}$  was found in Step 1B above.

The problem is now to choose  $\mathbf{W}_{ny}$ . One approach is to estimate it from the data using a preprocessing step. For example, one may do some preliminary regression and from this obtain an estimate of the noise.

A very simple approach, which is tested on some applications below, is to assume that the measurements have been scaled (for example, by the norm of the variation in each measurement), such that they have similar expected magnitudes. Thus, we use  $\mathbf{W}_{ny} = w_n\mathbf{Y}$  where  $w_n$  is a scalar.

```
noise = wn*eye(rx);
Xoptnoise = [Xopt noise];
[Hopt,loss]=soc_avg(Gy,Xoptnoise);
% etc.....
```

$w_n$  is a tuning parameter but a systematic approach is the following: Plot the singular values of  $\mathbf{X}_{opt}$  and select  $w_n$  such that the singular values of  $\mathbf{X}_{extended}$  follow the same trend.

### 2.9.3 Modification: Smoothing of Data

The loss regression method is based on the loss method where it is assumed that a model is available. When we are using data, then the model has to be extracted first (step 1), and this step of the method is not based on a rigorous approach to the final objective, which is to use the model for future predictions.

Therefore, it is likely that this step may be improved, One possible modification is suggested next, although it seems from the later numerical tests that it actually may have no effect.

#### 2.9.3.1 Smoothing of $y_1$

The above procedure (including step 1) assumes that all the noise is in the  $\mathbf{x} = \mathbf{y}$ . One way of dealing noise in  $y_1$  is to run through the procedure twice.

First, we go through the procedure (steps 1 and 2) and find the optimal  $\mathbf{H}_0$ .

Next, we go through the procedure again, but with a modified step 1A where use  $\mathbf{H}_0$  to estimate the smoothed (fitted) values of  $y_1$  that correspond to the measured  $\mathbf{X}$ ,

$$\mathbf{Y}_1^{\text{smooth}} = \mathbf{H}_0 \mathbf{X}$$

and then we use this smoothed data in the other steps,

$$\mathbf{Y}_{all} = [\mathbf{Y}_1^{\text{smooth}}, \mathbf{X}]$$

```

Y1smooth = Hoptloss*X;
% then redo step 1 and 2
Yallsmooth = [Y1smooth; X];
[u,s,v]=svd(Y1smooth);
% etc....

```

However, from the numerical examples this smoothing has no effect on the results, which in some sense is good, but on the other hand it does not offer any extra degrees of freedom.

### 2.9.4 Numerical Tests

The Matlab files for these tests can be found at the home page of S. Skogestad, and the commands are also listed in the Appendix.

The objective is to find  $\mathbf{H}$  (called beta in PLS Matlab) such that  $\mathbf{y} = \mathbf{H}\mathbf{x}$ . Note that PLS in all cases has an extra degree of freedom because it fits  $\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{h}_0$  where  $\mathbf{h}_0$  is nonzero. This may help for the fitting, but not for validation. In any case, all the data is centered, so we can assume  $\mathbf{h}_0$  is close to 0.

### 2.9.5 Test 1. Gluten Test Example from Harald Martens

Data: GlutenStarchNIR

1  $\mathbf{y}_1$  (gluten), 100  $\mathbf{x}$  (NIR absorbents), 100 data set).

This data has no noise on  $\mathbf{y}_1$ .

We first chose to use the 50 first data set for calibration and the 50 last for validation. Table 2.1 shows the fit  $\|\mathbf{Y}_1 - \mathbf{H}\mathbf{X}\|_F$  to calibration and validation data. The best result for the traditional methods is PCR with 33

**Table 2.1** Spectroscopic Example: Calibration and Validation data

Cal	Val	Method
0.0000	0.4758	Least squares (ls)
0.0738	0.3471	PCR-41 (tol=1.e-4)
0.1323	0.3142	PCR-33 (tol=2.e-4)
0.1890	0.4106	PCR-12 (tol=1.e-3)
0.0898	0.3608	PLS-8
0.0941	0.3445	PLS-7
0.1303	0.4987	PLS-6
0.0000	0.4735	min.loss
0.0000	0.4735	min.loss w/smooth y1
0.1442	0.3271	min.loss w/noise=5.e-4
0.1011	0.3115	min.loss w/noise=2.e-4

principal components (corresponding to a tolerance of 2e-4 when taking the pseudo inverse), which gives a validation fit of 0.3142. PLS can use much fewer components, but the fit is not as good (validation error is 0.3445) as PCR. As expected, the minimum loss method given perfect fit to the data and gives results identical to least squares. Thus, noise has to be added, and a noise level of 2.e-4 gives an error to the validation data (0.3115) which is even better than the best PCR

Use of smoothed  $y_1$ -data has no effect, which is expected as the fit to the calibration data is perfect.

**Table 2.2** Spectroscopic Example: Calibration and Validation data (with sets interchanged)

Cal	Val	Method
0.0000	0.6763	Least squares (ls)
0.1687	0.3873	PCR-12 (tol=5.e-4)
0.1652	0.2357	PLS-7
0.0000	0.6785	min.loss
0.1476	0.3798	min.loss w/noise=5.e-4

**Table 2.3** Spectroscopic Example: Calibration (99 measurements) and Validation data (1 rotating measurement)

Cal (avg.)	Val	Method
0.0000	3.0681	Least squares (ls) = min.loss
0.2689	0.3609	PCR (tol=5.e-4)
0.2769	0.3129	PLS-7
0.2471	0.3556	min.loss. w/noise=5.e-4

However, if we interchange the calibration and validation data set, then the results are very different; see Table 2.2. The best is now PLS-7 (val=0.2357), whereas minimum loss with noise is at 0.3798.

Finally, one data set was excluded at a time and used for validation. The average norm of the calibration fit and the norm of the validation fit for the 100 runs are given in Table 2.3. Again the PLS method is best (val=0.3129), whereas the minimum loss method with noise is the second best (val=0.2556).

### 2.9.6 Test 2. Wheat Test Example from Bjorn Alsberg (Kalivas, 1997)

Data: wheat spectra

2  $\mathbf{y}_1$ , 701  $\mathbf{x}$ , 100 data set.

The calibration data contains 50 measurements and the validation data set 50 measurements (specified in files I received).

The results are shown in Table 2.4. The loss method is the best (validation error 2.9 compared to 3.1 for the traditional methods), but the differences are small.

The difference between the methods is small for this test case and in this case the loss method gives better validation (2.9033) than least squares (3.1092) in spite of the fact that the calibration fit is perfect in both cases. This is a case where one would expect it to help to add artificial noise to the loss method. The reason is that we have 701  $\mathbf{x}$ 's but only 50 data sets for calibration, so the data would not be expected to contain sufficient information about the expected noise. However, the numerical results do not confirm this, and the fit gets worse when we add noise.

$\mathbf{Y}_1$  contains both the two y-variables (water and protein); presumably better fit can be obtained by fitting one at a time.

Table showing the fit  $\|\mathbf{Y}_1 - \mathbf{H}\mathbf{X}\|_F$  to calibration and validation data:

**Table 2.4** Wheat Spectra Calibration and Validation data

Cal.	Val.	Method
0.0000	3.1092	least squares (= PCR -50)
0.3136	3.0871	PCR-47 (tol=1.1 e-3)
0.0052	3.1052	PLS-35
0.0000	2.9033	min.loss
0.0000	2.9033	min.loss + noise=1.e-8
0.0069	3.1099	min.loss + noise=1.e-4

Again, smoothening of  $y_1$  has no effect, which again is not surprising since the fit was perfect.

### 2.9.7 Test 3. Our Own Example

The data contains 2  $\mathbf{y}_1$ , 7  $\mathbf{x}$ , 2  $\mathbf{d}$ .

$$\mathbf{G}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{G}_{d_1} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad g_{y_1} = [0.2034563] \quad g_{y_2} = [00.213489] \quad \mathbf{G}_y = [g_{y_1}^T, g_{y_2}^T] \quad g_{d_1} = [004568 - 9] \quad g_{d_2} = [00 - 3 - 55918] \quad \mathbf{G}_{y_d} = [g_{d_1}^T; g_{d_2}^T]$$

8 (or 32) noisy data set generated from ideal data  $[\mathbf{G}^y \mathbf{G}_d^y]$  (4 data sets) with 2 (or 8) different noise sets to get a total of 8 data sets. We here generate the data by adding noise to data from a model (“calibration set”) and the “validation set” is the noise-free data. To center the data I used opposite sign when the data was “repeated”. The noise was generated using randn command in Matlab.

It was found that when tested on a single example then almost any of the methods could be the winner. To avoid this effect, the comparison was run many times (with different random noise).

Table 2.5 shows the average value of  $\|\mathbf{Y}_1 - \mathbf{H}\mathbf{X}\|_F$  for 8 data sets after running 250 times. The best method is PCR (val=0.4931), PLS with 4 components (val=0.5137) is the best PLS. This is not surprising since we know that the data contains 4 directions. The new loss method is not doing so well in this case (val=1.055), which is not so surprising since with only 8 data sets there is limited information about the noise. Note that it is even worse than least squares (val=0.9985). As expected, the improvement by adding noise was significant (val=0.5850), but it is still not quite as good as PCR and PLS.

Surprisingly, smoothening of  $y_1$  had absolutely no effect in this case, even when I added noise on the  $\mathbf{y}_1$ -data (results not shown).

**Table 2.5** Our own example: 8 data sets in each run (average of 500 runs)

w/noise(cal)	no noise(val)	Method
0.2934	0.9985	LS (=PCR-8)
0.5496	0.4931	PCR (tol=1, most cases PCR-5)
0.5102	0.5137	PLS-4
0.3150	1.0552	loss method (no noise)
0.3150	1.0552	loss method (smooth y1)
0.4205	0.5850	loss method (noise=0.5)

Now, to show that the loss method does better when there is more data, Table 2.6 shows the average value of  $\|\mathbf{Y}_1 - \mathbf{H}\mathbf{X}\|_F$  for 32 data sets after running 300 times.

**Table 2.6** Our own example: 32 data sets in each run (average of 300 runs)

w/noise(cal)	no noise(val)	Method
1.4826	0.3560	LS (=PCR-8)
1.4970	0.3560	PCR (most cases PCR-6)
1.5256	0.3698	PLS-4
1.7700	0.2690	loss method (no noise)
1.7700	0.2690	loss method (smooth y1)
1.7703	0.2687	loss method (noise=0.5)

The loss method is the winner (val = 0.269) in spite of the fact that it has no “tuning” parameters.

Here there is little effect of adding artificial noise with the loss method, presumably because we have enough data.

### 2.9.8 Comparison with Normal Least Squares

Normal least square solution. Problem: Find  $\mathbf{H}$  such that the magnitude of  $\|\mathbf{Y}_1 - \mathbf{H}\mathbf{Y}\|_2$  is minimized for the given set of data for  $\mathbf{Y}_1$  and  $\mathbf{Y}$ .

Solution:  $\mathbf{H} = \mathbf{Y}_1 \text{pinv}(\mathbf{Y})$ . This minimizes  $\|\mathbf{Y}_1 - \mathbf{H}\mathbf{Y}\|_2$  and, for cases where this  $\mathbf{H}$  is not unique, minimizes  $\|\mathbf{H}\|_2$ .

This is the same as finding  $\mathbf{H}$  to minimize  $\|[\mathbf{I} \quad \mathbf{H}]\mathbf{Y}_{all}\|$ , which we know is not the optimal solution

*Proof.* The data matrix is  $\mathbf{Y}_{all} = [\mathbf{Y}_1; \mathbf{Y}]$ . Assume that we seek  $\mathbf{H}_{all}$  to minimize  $\|\mathbf{H}_{all}\mathbf{Y}_{all}\|_2$ . We have degrees of freedom in  $\mathbf{H}_{all}$ , so we set  $\mathbf{H}_{all} = [\mathbf{I} \quad -\mathbf{H}]$ . Then we want to minimize  $\|\mathbf{H}_{all}[\mathbf{Y}_1; \mathbf{Y}]\|_2 = \|-\mathbf{Y}_1 + \mathbf{H}\mathbf{Y}\|$  the best solution is given by the pseudo inverse,  $\mathbf{H} = \mathbf{Y}_1 \text{pinv}(\mathbf{Y})$  which is the usual least square solution.  $\square$

So why is least squares not optimal?

The “problem” (objectives function) for normal least squares is to get the best match of the available data, ie., minimize  $\|\mathbf{Y}_1 - \mathbf{H}\mathbf{X}\|$ , and it does not consider how the estimate  $\mathbf{y}_1 = \mathbf{H}\mathbf{x}$  is going to be used in the future.

So why is the loss approach expected to be better?

In the loss approach, the problem is:

Given that  $\hat{\mathbf{y}}_1 = \mathbf{H}\mathbf{x}$ , find an optimal  $\mathbf{H}$  such that the average magnitude of  $\|\mathbf{y}_1 - \hat{\mathbf{y}}_1\|_2$  is minimized for the expected future  $\mathbf{d}s$  and  $\mathbf{n}^y$ s (which are assumed 2-norm bounded).

Here, we use the data to obtain  $\mathbf{Y} = \mathbf{Y}_{opt}$ ,  $\mathbf{G}^y$  and  $\mathbf{G}_1$  (step 1). This step may possibly be improved but it seems reasonable.

The main advantage is that in step 2, we obtain the estimate  $\mathbf{y}_1 = \mathbf{H}\mathbf{y}$  that will work best “on average” for the expected disturbances and measurement noise (as are indirectly given by the data in  $\mathbf{Y}_{opt}$ , that is, we consider the future use of the estimator and not just fitting of the data).

## 2.10 Discussion

### 2.10.1 Gradient Information

How can we use the proposed approach in practice, for example, to find the optimal policy for a marathon runner. We need to be able to distinguish between “optimal data” ( $\mathbf{Y}^{opt}$ ) and “nonoptimal data” to find  $\mathbf{G}^y$  and  $\mathbf{G}_1$ . A simple approach is to set  $\mathbf{y}_1 = \mathbf{J}_u$  (that is, we want to estimate the gradient). Then we know that optimal data corresponds to  $\mathbf{y}_1 = \mathbf{J}_u = 0$  and we can do exactly as above (Data approach 1 or 2). However, note that this means that we need some non-optimal data where we know the value of  $\mathbf{J}_u$ .

### 2.10.2 Relationship to NCO tracking

Finally, some ideas related to NCO tracking.

The NCO-idea is to set the Gradient= 0. An important difference compared to the proposed loss approach, is that in NCO-tracking one tries to find an expression for  $\mathbf{u}$  (“open-loop implementation” similar to deadbeat control).

On the other hand in the loss method (self-optimizing control), we “stop” when we have the expression for the gradient  $\mathbf{c} = \mathbf{J}_u$ . the implementation to find  $\mathbf{u}$  that gives  $\mathbf{J}_u = 0$  is by feedback control!

From (2.10) and (2.11) we see that we want to minimize  $(\mathbf{u} - \mathbf{u}_{opt})$  or  $\mathbf{J}_u$  (but weighted by  $\mathbf{J}_{uu}$ ). Recall here that in the full- $\mathbf{H}$  case,  $\mathbf{J}_{uu}$  is not needed. Still, it remains unclear if this means that we can just minimize  $\|\mathbf{J}_u\|$ ?

Another problem with NCO idea to “Control gradient to zero” is that this is not really possible since gradient can not be measured. Thus, it needs to be estimated. For the case with no noise the estimate  $\mathbf{J}_u$  is same as “nullspace method”, so  $\mathbf{c} = \mathbf{H}\mathbf{y} = \mathbf{J}_u!$

For noisy case not so clear, but may as well use  $\mathbf{c} = \mathbf{H}\mathbf{y}$ .

## 2.11 Appendix

```

1
2 % This is file matlab-test-cases.m
3 % Load data from C:\Documents and Settings\skoge\My Documents\
  MATLAB
4
5 % -----
6 % Test case 1. Martens data:
7 load GlutenStarchNIR
8 % X: Variablene 13-112 er NIR absorbanser (log(1/T) fra
  850-1048 nm).
9 % y1: Variabel 4 er kvantitativ analytt-konsentrasjon (gluten),
10 % Matrix: 100 rows 112 cols: regular MATLAB matrix
11 % VarLabels: 112 rows 17 cols: MATLAB character string
12 % ObjLabels: 100 rows 7 cols: MATLAB character string
13 Yall=Matrix';
14 XX = Yall(13:112,:);
15 YY = Yall(4,:);
16 X = XX(:,1:50);
17 X0= XX(:,51:100);
18 Y1= YY(:,1:50);
19 Y10=YY(:,51:100);
20
21 % repeated Martens
22 % 100 times where I take out data each time
23 %for nsis=1:98
24 %X = XX(:,[1:nsis nsis+2:100]); X0 = XX(:,nsis+1);
25 %Y1 = YY(:,[1:nsis nsis+2:100]); Y10 = YY(:,nsis+1);
26 % Two end cases are handled separately
27 %nsis=0
28 %X = XX(:,[nsis+2:100]); X0 = XX(:,nsis+1);
29 %Y1 = YY(:,[nsis+2:100]); Y10 = YY(:,nsis+1);
30 %nsis=99
31 %X = XX(:,[1:nsis]); X0 = XX(:,nsis+1);
32 %Y1 = YY(:,[1:nsis]); Y10 = YY(:,nsis+1)
33

```



```

34 % Jump to.. MATLAB code starts here
35
36 % -----
37 % Test case 2.
38 % Bjørn Alsberg data (epost 15.11 2010)
39 load wheat_spectra
40 %Your variables are:
41 %X      Y      idxcal  idxval  moist  protein
42 Xwheat=X'; Ywheat=Y'; % 2 y's, 701 x's, 100 data set
43 X =Xwheat(:,idxcal); Y1 =Ywheat(:,idxcal); % 50 calibration
    sets
44 X0=Xwheat(:,idxval); Y10=Ywheat(:,idxval); % 50 validation sets
45 %X =Xwheat(:,idxval); Y1 =Ywheat(:,idxval); % 50 calibration
    sets - switched
46 %X0=Xwheat(:,idxcal); Y10=Ywheat(:,idxcal); % 50 validation
    sets - switched
47
48
49 % Jump to.. MATLAB code starts here
50
51 % -----
52 % Test case 3. Own example
53
54 G1 = [1 0; 0 1]
55 Gd1= [0 0; 0 0]
56 gy1 =[0.2 0 3 4 5 6 3]
57 gy2 =[0 0.2 1 3 4 8 9]
58 Gy = [gy1',gy2']
59 gd1 =[0 0 4 5 6 8 -9]
60 gd2 =[0 0 -3 -5 5 9 18]
61 Gyd = [gd1',gd2']
62
63 Y10 = [G1 Gd1]
64 X0 = [Gy Gyd]
65
66 Y100=Y10;
67 na=0; aa=a*0;
68
69 % Run repeatedly from here for several cases
70
71 Y10=Y100;
72
73 % 8 data sets
74 Noise = 0.5*randn(7,8)
75 X = [X0 -X0] + Noise

```

```

76 Y1 = [Y10 -Y10] % use Y10 and -Y10 to get centered data
77
78 %%32 data sets
79 %X = [X0 -X0]
80 %Noise = 0.5*randn(7,32)
81 %X = [X X X X] + Noise
82 %Y1 = [Y10 -Y10] % use Y10 and -Y10 to get centered data
83 %Y1 = [Y1 Y1 Y1 Y1]
84 %%NoiseY= 0.0*randn(2,32) % with 2 y1's
85 %%Y1 = [Y1 Y1 Y1 Y1] + NoiseY
86 % 100 times where I take out data each time
87
88 %-----
89 % MATLAB code starts here
90
91 % Least squares (= PCR with default tol)
92 method1='LS';
93 Hls = Y1*pinv(X);
94 res=Hls*X;
95 res0=Hls*X0;
96 a11=norm(res-Y1);
97 a12=norm(res0-Y10);
98
99 % PCR (vary tol)
100 % PCR: To find cut-off to find no. of components, semilogy(svd(
    X))
101 method6='PCR';
102 tol=1.e-4
103 Hpcr = Y1*pinv(X,tol);
104 npcr=rank(pinv(X,tol)) % no. of components used in PCR
105 res=Hpcr*X;
106 res0=Hpcr*X0;
107 a61=norm(res-Y1);
108 a62=norm(res0-Y10);
109
110 % Weighted least squares (to get relative noise assumption
    rather than additive noise)
111 method5='weightedLS';
112 [rx,cx]=size(X);
113 mag=[];
114 for i = 1:rx
115 mag=[mag norm(X(i,:))]; % the magnitudes are about 0.8
116 end
117 Xs = inv(diag(mag))*X;
118 Hlss1 = Y1*pinv(Xs);

```

```

119 Hlss = Hlss1*inv(diag(mag));
120 res=Hlss*X;
121 res0=Hlss*X0;
122 a51=norm(res-Y1) ;
123 a52=norm(res0-Y10) ;
124
125 % pls
126 nppls=35
127 %nppls=npcr
128 [XL,y1,XS,YS,beta,PCTVAR] = plsregress(X',Y1',nppls);
129 % Note that PLS has an additional bias/centering parameter.
130 yfit = [ones(size(X',1),1) X']*beta;
131 yfit0 = [ones(size(X0',1),1) X0']*beta;
132 a71=norm(yfit-Y1');
133 a72=norm(yfit0-Y10');
134 method7='UUUUPLS';
135
136 % Loss method
137 % step 1
138 Yall = [Y1; X];
139 [u,s,v]=svd(Y1);
140 Yall1 = Yall*v;
141 [r1,c1]=size(Yall);
142 [r2,c2]=size(Y1);
143 ny=r2;
144 G1=Yall1(1:ny,1:ny);
145 Gy=Yall1(ny+1:r1,1:ny);
146 Xopt = Yall1(ny+1:r1,ny+1:c1);
147
148 % step 2,
149 %Hopt = (pinv(Xopt*Xopt')*Gy)'; %analytical expression
150 [Hopt,loss]=soc_avg(Gy,Xopt);
151 D=Hopt*Gy*inv(G1); %
152 Hopt4=inv(D)*Hopt;
153 resb=Hopt4*X;
154 resb0=Hopt4*X0 ;
155 a41=norm(resb-Y1) ;
156 a42=norm(resb0-Y10) ;
157 method4='UUUUloss_';
158
159 % NEW modified loss method for case with noise on Y: redo step
    1.
160 % New estimate of Y1.
161 Y1smooth = Hopt4*X;
162 % then redo step 1 and 2

```

```

163 Yallsmooth = [Y1smooth; X];
164 [u,s,v]=svd(Y1smooth);
165 Yallsmooth = Yall*v;
166 G1smooth=Yallsmooth(1:ny,1:ny);
167 Gysmooth=Yallsmooth(ny+1:r1,1:ny);
168 Xsmooth = Yallsmooth(ny+1:r1,ny+1:c1);
169 % step 2
170 [Hopt,loss]=soc_avg(Gysmooth,Xsmooth);
171 D=Hopt*Gysmooth*inv(G1smooth); %
172 Hopt10=inv(D)*Hopt;
173 resa=Hopt10*X;
174 resa0=Hopt10*X0 ;
175 a101=norm(resa-Y1) ;
176 a102=norm(resa0-Y10) ;
177 method10='llllosslw/smooth';
178
179 % loss method: add artificial noise weights
180 noisemag=2.e-4
181 %noisemag=tol
182 % Noise Alt.1 just additive noise:
183 noise = noisemag*eye(rx);
184 Xoptnoise = [Xopt noise];
185 % Noise Alt2. Add noise proportional to variation in each
    output
186 % [rx,cx]=size(Xopt);
187 %mag=[];
188 %for i = 1:rx
189 %mag=[mag norm(Xopt(i,:))]; % the magnitudes are about 0.8
190 %end
191 %noise = noisemag*diag(mag);
192 %Xoptnoise = [Xopt noise];
193
194 % step 2- with artificial noise
195 % semilogy(svd(X))
196 % semilogy(svd([X X0])
197 [Hopt,loss]=soc_avg(Gy,Xoptnoise);
198 D=Hopt*Gy*inv(G1); %
199 Hopt9=inv(D)*Hopt;
200 resb=Hopt9*X;
201 resb0=Hopt9*X0 ;
202 a91=norm(resb-Y1) ;
203 a92=norm(resb0-Y10) ;
204 method9='lllllosslw/noise';
205
206 % Summary of results

```

```
207 methods=[method1 method5 method6 method7 method4 method10
        method9]
208 a =[a11 a12; a51 a52; a61 a62; a71 a72; ; a41 a42; a101 a102;
        a91 a92]’
209
210 % For repeated case 1 and case 3:
211 na=na+1
212 aa=aa+a;
213 aaa=aa/na
214 % For repeated case 1
215 end
```

---

**Acknowledgements** The authors are pleased to acknowledge the financial support from the NIL-I-007-d project and the Norwegian Research Council

## References

- Alstad V, Skogestad S (2007) Null space method for selecting optimal measurement combinations as controlled variables. *Ind Eng Chem Res* 46:846–853
- Alstad V, Skogestad S, Hori E (2009) Optimal measurement combinations as controlled variables. *Journal of Process Control* 19(1):138–148
- Halvorsen IJ, Skogestad S, Morud JC, Alstad V (2003) Optimal selection of controlled variables. *Ind Eng Chem Res* 42
- Hori ES, Skogestad S, Alstad V (2005) Perfect steady-state indirect control. *Industrial & Engineering Chemistry Research* 44(4):863–867
- Kariwala V, Cao Y (2009) Bidirectional branch and bound for controlled variable selection. part ii: Exact local method for self-optimizing control. *Computers and Chemical Engineering* 33:1402–1414
- Kariwala V, Cao Y, Janardhanan S (2008) Local self-optimizing control with average loss minimization. *Ind Eng Chem Res* 47:1150–1158
- Yelchuru R, Skogestad S (2010) Miquip formulation for optimal controlled variable selection in self optimizing control. In: *PSE Asia, 2010. July 25-28, Singapore*

*Comments – Remarks*

*Comments – Remarks*

*Comments – Remarks*



*Comments – Remarks*

# Chapter 3

## Measurement polynomials as controlled variables – Exercises

Johannes Jäschke and Sigurd Skogestad

**Abstract** In this chapter we present two exercises for finding controlled variables, which are polynomials in the measurements. Detailed solutions and maple source code are included so that the reader can easily follow the procedure.

### 3.1 Introduction

To illustrate concepts from described in the textbook, we present some small problem and go through the solution step by step. The reader is encouraged to experiment on his own to understand the ideas better.

For solving the CSTR case study, the `multires` package is required<sup>1</sup>

### 3.2 Simple exercise

**Exercise 3.1 (Cox 1992).** Check whether the two polynomials  $f_1 = 2x^2 + 3x + 1$  and  $f_2 = 7x^2 + x + 3$  have a common root in  $\mathbb{C}$ .

**Solution 3.1.** The resultant is the determinant of the Sylvester matrix:

---

Johannes Jäschke  
Department of Chemical Engineering, NTNU Trondheim, Norway, e-mail:  
[jaschke@chemeng.ntnu.no](mailto:jaschke@chemeng.ntnu.no)

Sigurd Skogestad  
Department of Chemical Engineering e-mail: [skoge@chemeng.ntnu.no](mailto:skoge@chemeng.ntnu.no)

<sup>1</sup> The software can be downloaded at [www-sop.inria.fr/galaad/software/multires](http://www-sop.inria.fr/galaad/software/multires)

$$\text{Res}(f_1, f_2) = \det \left( \begin{bmatrix} 2 & 0 & 7 & 0 \\ 3 & 2 & 1 & 7 \\ 1 & 3 & 3 & 1 \\ 0 & 1 & 0 & 3 \end{bmatrix} \right) = 153 \neq 0 \quad (3.1)$$

There exist no common root since the resultant is nonzero.

Maple code for the Sylvester matrix example

```

1 with(LinearAlgebra):
2 f1 := 2*x^2+3*x+1;
3 f2 := 7*x^2+x+3;
4 Syl := SylvesterMatrix(f1,f2);
5 Res := Determinant(Syl);

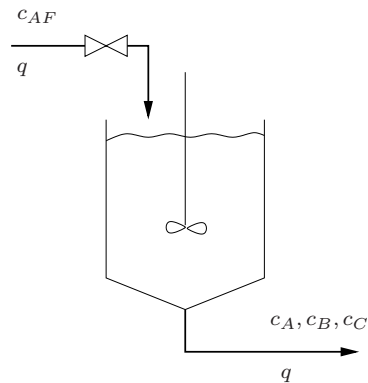
```

### 3.3 Isothermal CSTR Case Study

Consider a CSTR as in Figure 3.1, with a feed stream containing component  $A$  and with two first order chemical reactions,



Of the products formed,  $B$  is the desired product, while  $C$  is an undesired side product. The manipulated variable is the feed stream  $q$ , which can be adjusted to achieve profitable performance. The operational objective is to maximize the concentration of the desired product.



**Fig. 3.1** Isothermal CSTR

It is assumed that the unmeasured disturbances are  $k_1$  and  $k_2$ , and all other variables are known except of  $c_B$ , which is assumed difficult to measure. The unmeasured variables are summarized in Table 3.1, and all measurements

Symbol	Description
$k_1$	Reaction constant for reaction 1
$k_2$	Reaction constant for reaction 2
$c_B$	Concentration of desired product

**Table 3.1** Unmeasured variables

and known parameters are shown in Table 3.2. The task is to find a con-

Symbol	Description	Type	Value	Unit
$q$	Feed flow rate	Known input	varying	$\text{m}^3/\text{min}$
$c_A$	Outlet concentration $A$	Measurement	varying	$\text{kmol}/\text{m}^3$
$c_C$	Outlet concentration $C$	Measurement	varying	$\text{kmol}/\text{m}^3$
$V$	Tank volume	Known parameter	0.9	$\text{m}^3$
$c_{AF}$	Feed concentration $A$	Known parameter	10.0	$\text{kmol}/\text{m}^3$
$c_{BF}$	Feed concentration $B$	Known parameter	0.0	$\text{kmol}/\text{m}^3$
$c_{CF}$	Feed concentration $C$	Known parameter	0.0	$\text{kmol}/\text{m}^3$

**Table 3.2** Known variables: Inputs, measurements and parameters

trolled variable which can be controlled using the total flow rate, and which maximizes the desired concentration.

Subtasks:

1. Set up the steady state component balances
2. Set up the optimization problem
3. Write the optimality conditions
4. Calculate the reduced gradient
5. Eliminate the unknown Variables

## 3.4 Solution

### 3.4.1 Component Balance

We do a The steady state component balances for the system read:

$$\begin{aligned}
 g_1 &= qc_{AF} - qc_A - k_1 c_A V = 0 \\
 g_2 &= qc_{BF} - qc_B + k_1 c_A V - k_2 c_B V = 0 \\
 g_2 &= qc_{CF} - qc_C + k_2 c_B V = 0
 \end{aligned} \tag{3.3}$$

### 3.4.2 Optimization Problem

The objective function is

$$J = -c_B, \quad (3.4)$$

which we want to minimize subject to the process model:

$$\begin{aligned} \min J \\ \text{s.t.} \\ g_1 = 0 \\ g_2 = 0 \\ g_3 = 0 \end{aligned} \quad (3.5)$$

### 3.4.3 Optimality Conditions

We write the Lagrangian

$$\mathcal{L} = J(\mathbf{z}) + \lambda^T g(\mathbf{z}), \quad (3.6)$$

where  $\mathbf{z} = (c_A, c_B, c_C, q)^T$ . The first order optimality conditions are then

$$\begin{aligned} \nabla_{\mathbf{z}} J(\mathbf{z}) + \nabla_{\mathbf{z}} g(\mathbf{z}) &= 0 \\ g(\mathbf{x}) &= 0 \end{aligned} \quad (3.7)$$

Next we calculate the null-space of the constraints  $\mathbf{N} = [n_1, n_2, n_3, n_4]^T$  with

$$n_1 = (c_{AF} - c_A)/(q + k_1V) \quad (3.8)$$

$$n_2 = -\frac{-qc_{BF} - c_{BF}k_1V + qc_B + c_Bk_1V - k_1Vc_{AF} + k_1c_AV}{(q + k_2V)(q + k_1V)} \quad (3.9)$$

$$\begin{aligned} n_3 = & -\left(-c_{CF}q^2 - c_{CF}qk_1V - c_{CF}k_2Vq - c_{CF}k_2V^2k_1 + c_Cq^2 + c_Cqk_1V \right. \\ & + c_Ck_2Vq + c_Ck_2V^2k_1 - k_2Vqc_{BF} - k_2V^2c_{BF}k_1 + k_2Vqc_B + k_2V^2c_Bk_1 \\ & \left. - k_2V^2k_1c_{AF} + k_2V^2k_1c_A\right) / ((q + k_2V)(q + k_1V)q) \end{aligned} \quad (3.10)$$

$$n_4 = 1 \quad (3.11)$$

Eliminate Lagrangian multipliers using the null space, we write the optimality conditions:

$$\begin{aligned} c^v = \mathbf{N}(\mathbf{z})^T \nabla_{\mathbf{z}} J(\mathbf{z}) &= 0 \\ g(\mathbf{z}) &= 0 \end{aligned} \quad (3.12)$$

Since we control  $c^v$  to zero, we need to consider only the numerator, which is

$$Num(c^v) := -qc_{BF} - c_{BF}k_1V + qc_B + c_Bk_1V - k_1Vc_{AF} + k_1c_AV; \quad (3.13)$$

This expression cannot be used for control yet, because it contains unknown variables. These have to be eliminated in the next step.

#### 3.4.4 Eliminating Unknowns $k_1, k_2$ and $c_B$

We use the package `multires` to construct the matrix for the toric resultant

$$M = \begin{bmatrix} -qc_{BF} & q & -c_{BF}V - Vc_{AF} + c_AV & V & 0 & 0 & 0 \\ qc_{BF} & -q & c_AV & 0 & 0 & -V & 0 \\ qc_{AF} - qc_A & 0 & -c_AV & 0 & 0 & 0 & 0 \\ 0 & 0 & qc_{BF} & -q & c_AV & 0 & -V \\ 0 & 0 & qc_{AF} - qc_A & 0 & -c_AV & 0 & 0 \\ qc_{CF} - qc_C & 0 & 0 & 0 & 0 & V & 0 \\ 0 & 0 & qc_{CF} - qc_C & 0 & 0 & 0 & V \end{bmatrix} \quad (3.14)$$

#### 3.4.5 The Determinant

The (factorized) determinant of  $M$  is

$$\tilde{c} = q^3c_AV^4(c_{AF}c_A + c_{AF}c_{CF} - c_{AF}c_C - c_A^2) \quad (3.15)$$

We see that the pre-factor  $q^3c_AV^4$  is nonzero under operation, so the condition for optimal operation is only the last factor:

$$c = c_{AF}c_A + c_{AF}c_{CF} - c_{AF}c_C - c_A^2 = 0 \quad (3.16)$$

Now we have ended up with a controlled variable combination which contains only known variables. It may be confirmed by the reader that controlling  $c$  to zero leads to the same solution as solving the optimization problem (3.5).

### 3.4.6 Maple Code

The maple code for the CSTR example is given below

Maple code for the CSTR example

```

1 #####
2 ## This is the file mapleCSTR.mpl
3 ## Simple CSTR
4 ## Johannes Jaeschke"
5 ## Nov. 2010
6 #####
7
8 with(LinearAlgebra):with(VectorCalculus):
9 # Define the cost to minimize
10 J :=-cB;
11
12 # Setting up the constraints
13 g1 :=q*cAF - q*cA - k1*cA*V;
14 g2 :=q*cBF - q*cB + k1*cA*V - k2*cB*V;
15 g3 :=q*cCF - q*cC          + k2*cB*V;
16 g :=[g1,g2,g3];
17
18 # Derive to obtain first order optimality constraints
19 Jac := Jacobian(g,[cA,cB,cC,q]);
20 gradT := Jacobian([J],[cA,cB,cC,q]);
21
22 # Calculate the null space of the constraints
23 N := NullSpace(Jac): N := N[1];
24
25 # Transpose and mutlitypy
26 G := Transpose(gradT):
27 NT := Transpose(N):
28 # The reduced gradient is then:
29 Cv := VectorMatrixMultiply(NT,G):
30 cv := simplify(numer(Cv[1])): # pick only numerator
31
32 # Unknown variables to be eliminated
33 read("multires.mpl"):
34 varlist := [k2,k1,cB]; #unknown variables
35 polylist:= [cv,g1,g2,g3]; #optimality conditions
36 BigMat := spresultant(polylist,varlist); #Construct resultant matrix
37 LargeMat:= det(BigMat); # Calculate determinant
38 c := factor(LargeMat); # factorize the CV
39 save Jac,grad,N,Cv,cv,c, "invariant"; # save results

```

*Comments – Remarks*



*Comments – Remarks*

*Comments – Remarks*

*Comments – Remarks*

# Chapter 4

## Multi-Parametric Toolbox

Michal Kvasnica

**Abstract** This chapter introduces the Multi-Parametric Toolbox (MPT) for MATLAB, which is a freely-available tool for Model Predictive Control and computational geometry. MPT simplifies and automates many tasks a control engineer has to go through when designing and validating optimal control laws based on the MPC principle. The toolbox provides easy-to-use access to various tasks ranging from modeling of dynamical systems, through control synthesis, up to verification and code deployment. The purpose of this chapter is twofold. First, the toolbox is introduced and its functionality is explained by means of simple, illuminating examples. Then, a set of advanced exercises is provided for the reader to practice and master their skills.

### 4.1 Multi-Parametric Toolbox

Optimal control of constrained linear and piecewise affine systems has garnered great interest in the research community due to the ease with which complex problems can be stated and solved. The aim of the *Multi-Parametric Toolbox* (MPT) is to provide efficient computational means to obtain feedback controllers for these types of constrained optimal control problems in a MATLAB programming environment. As the name of the tool hints, it is mainly focused on calculation of feedback laws in the *parametric* fashion in which the feedback law takes a form of a PWA look-up table. But the toolbox is also able to formulate and solve MPC problems on-line in the receding horizon fashion, i.e. by solving the optimization problem for a particular value of the initial condition at each time step.

---

Michal Kvasnica  
Faculty of Chemical and Food Technology, Slovak University of Technology  
in Bratislava, e-mail: [michal.kvasnica@stuba.sk](mailto:michal.kvasnica@stuba.sk)

In short, the Multi-Parametric Toolbox can be described as being a free MATLAB toolbox for design, analysis and deployment of MPC-based control laws for constrained linear, nonlinear and hybrid systems. Efficiency of the code is guaranteed by the extensive library of algorithms from the field of computational geometry and multi-parametric optimization. The toolbox offers a broad spectrum of algorithms compiled in a user friendly and accessible format: starting from modeling systems which combine continuous dynamics with discrete logic (hybrid systems), through design of control laws based on different performance objectives (linear, quadratic, minimum time) to the handling of systems with persistent additive and polytopic uncertainties. Users can add custom constraints, such as polytopic, contraction, or collision avoidance constraints, or create custom objective functions. Resulting optimal control laws can either be embedded into control applications in the form of a C code, or deployed to target platforms using Real Time Workshop.

MPT can also be viewed as a unifying repository of hybrid systems design tools from international experts utilizing state-of-the-art optimization packages. The list of included software packages includes packages for linear programming (CDD, GLPK), quadratic programming (CLP), mixed-integer linear programming (GLPK), and semi-definite programming (SeDuMi). In addition, MPT ships with a dedicated solver for computing projections of convex polytopes, called ESP, a boolean optimization package ESPRESSO, as well as with the HYSDEL modeling language.

The main factor which distinguishes this toolbox from other alternatives is the big emphasis on efficient formulation of the problems which are being solved. This means that the toolbox provides implementation of novel control design and analysis algorithms, but also offers the user an easy way to use them without the need to be an expert in the respective fields. MPT aims at providing tools which can be used in the whole chain of the process of successful control design. It allows users not only to design optimization-based controllers, but also to formally verify that they behave as desired, investigate the behavior of the closed-loop system, and to post-process the resulting feedback laws in order to simplify them without losing prescribed design properties.

#### *4.1.1 Download and Installation*

MPT is freely available for download from the project web site at <http://control.ee.ethz.ch/~mpt>. After download, follow the installation instructions and add all MPT's sub-directories to MATLAB path. To start off, check your MPT installation by running the following command:

```
>> mpt_init
```

which should provide the output similar to this one:

```
looking for available solvers...
```

```
MPT toolbox 2.6.3 initialized...
```

```

      LP solver: CDD Criss-Cross
      QP solver: quadprog
      MILP solver: GLPKCC
      MIQP solver: YALMIP
Vertex enumeration: CDD
```

Any questions, suggestions and/or bug reports should be communicated to [mpt@control.ee.ethz.ch](mailto:mpt@control.ee.ethz.ch).

## 4.2 Computational Geometry in MPT

### 4.2.1 Polytopes

First we introduce the computational geometry features of MPT. MPT contains a rich library for manipulating convex geometrical objects known as *polytopes*.

**Definition 4.1 (Convex Set).** A set  $\mathcal{C} \subseteq \mathbb{R}^n$  is convex if the line segment connecting any two points in  $\mathcal{C}$  lies entirely in  $\mathcal{C}$ , i.e.  $\theta x_1 + (1 - \theta)x_2 \in \mathcal{C}$  for all  $x_1, x_2 \in \mathcal{C}$  and  $\theta \in \mathbb{R}$ ,  $0 \leq \theta \leq 1$ .

**Definition 4.2 (Polyhedron).** Convex set represented as the intersection of a finite number of closed half-spaces  $a_i^T x \leq b_i$ , i.e.  $\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax \leq b\}$  with

$$A = \begin{bmatrix} a_1^T \\ \vdots \\ a_m^T \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \quad (4.1)$$

is called a polyhedron.

**Definition 4.3 (Polytope).** Polytope is a bounded polyhedron.

In general, polytopes can be described in two possible representations:

1. The  $\mathcal{H}$ -representation, i.e.  $\mathcal{P} = \{x \mid Ax \leq b\}$ , where the input data  $A$  and  $b$  are matrices representing the defining half-spaces  $a_i^T x \leq b_i$ , or
2. the  $\mathcal{V}$ -representation, i.e.  $\mathcal{P} = \text{convh}\{x_1, \dots, x_m\}$ , where  $\text{convh}$  denotes the convex hull operator, i.e.  $\text{convh}\{x_1, \dots, x_m\} = \{x \mid x = \lambda_1 x_1 + \dots + \lambda_m x_m, 0 \leq \lambda_i \leq 1, \sum \lambda_i = 1\}$ . Here, the input data are the vertices  $x_1, \dots, x_m$  of the polytope.

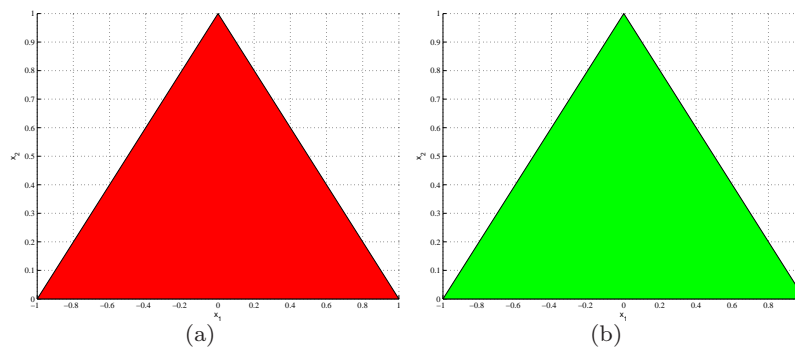
In MPT, polytopes are constructed using the `polytope` constructor<sup>1</sup>:

```
% define a triangle using inequalities
% x1 + x2 <= 1
% -x1 + x2 <= 1
%      -x2 <= 0
A = [1 1; -1 1; 0 -1];
b = [1; 1; 0];
TH = polytope(A, b)

% define the triangle using vertices
x1 = [1; 0]; x2 = [-1; 0]; x3 = [0; 1];
TV = polytope([x1 x2 x3]') % notice the transpose!
```

Calling the constructor returns a polytope object which can be processed in multiple ways. Let us start by simple plots:

```
% plot the first polytope in a new window using red color
figure; plot(TH, 'r');
% plot the first polytope in a new window using green color
figure; plot(TV, 'g')
```



**Fig. 4.1** Plot of polytopes TH (left) and TV (right).

The output of these commands is shown in Figure 4.1. It appears that the two polytopes are equal. But how to check this in an automated fashion? Easily, use the standard equivalence `==` MATLAB operator:

```
% are they really equal?
TH == TV
```

```
ans =
```

---

<sup>1</sup> See `help polytope` for more information.

1

Other supported operators include  $P \leq Q$ , which checks whether the polytope  $P$  is a subset of  $Q$ , and  $P \geq Q$  which checks whether  $P$  is a superset of  $Q$ .

Another frequent operation is to check whether a given point is contained in the polytope. The `isinside` function provides a YES/NO answer to this question:

```
x = [0.5; 0.5];
% is this point contained in our first triangle?
isinside(TH, x)
ans =

    1

% of course, the function also works for V-polytopes:
isinside(TV, x)
ans =

    1

% now try a point outside the triangle:
x = [10; 10];
isinside(TV, x)
ans =

    0
```

Regardless of whether the polytope was constructed using inequalities or vertices, it is always possible to extract both the  $\mathcal{H}$ - as well as the  $\mathcal{V}$ -representation. To compute the extremal vertices of a given polytope, use the `extreme` function (see also `help polytope/extreme`):

```
E = extreme(TH)

ans =

    0    1
    1    0
   -1    0
```

To extract the  $\mathcal{H}$ -representation, use the `double` command:

```
[A, b] = double(TV)

A =

    0.7071    0.7071
   -0.7071    0.7071
```



```

      0   -1.0000
b =
    0.7071
    0.7071
     0

```

Notice that in this output all rows of  $A, b$  have been automatically normalized such that the 2-norm of each row is equal to one to improve numerics.

### 4.2.2 Polytope Arrays

Instances of the `polytope` object can be concatenated into arrays. Currently, only one-dimensional arrays are supported by MPT and it does not matter if the elements are stored row-wise or column-wise. An array of polytopes is created using standard MATLAB concatenation operators `[,]`, e.g.  $A = [B, C, D]$ .

It does not matter whether the concatenated elements are single polytopes or polyarrays. To illustrate this, assume that we have defined polytopes  $P1, P2, P3, P4, P5$  and polyarrays  $A = [P1 P2]$  and  $B = [P3 P4 P5]$ . Then the following polyarrays  $M$  and  $N$  are equivalent:

$$M = [A B]$$

$$N = [P1 P2 P3 P4 P5]$$

Individual elements of a polyarray can be obtained using the standard referencing (`i`) operator, i.e.

$$P = M(2)$$

will return the second element of the polyarray  $M$  which is equal to  $P2$  in this case. More complicated expressions can be used for referencing:

$$Q = M([1, 3:5])$$

will return a polyarray  $Q$  which contains first, third, fourth and fifth element of polyarray  $M$ .

If the user wants to remove some element from a polyarray, he/she can use the referencing command as follows:

$$M(2) = []$$

which will remove the second element from the polyarray  $M$ . Again, multiple indices can be specified, e.g.

$$M([1 3]) = []$$

will erase first and third element of the given polyarray. If some element of a polyarray is deleted, the remaining elements are shifted towards the start of the polyarray. This means that, assuming  $N = [P1 P2 P3 P4 P5]$ , after

```
N([1 3]) = []
```

the polyarray  $N = [P2\ P4\ P5]$  and the length of the array is 3. No empty positions in a polyarray are allowed. Similarly, empty polytopes are not being added to a polyarray.

A `polyarray` is still a polytope object, hence all functions which work on polytopes support also polyarrays. This is an important feature mainly in the geometric functions. Length of a given `polyarray` is obtained by

```
l = length(N)
```

A polyarray can be flipped by the following command:

```
Nf = fliplr(N)
```

i.e. if  $N = [P1\ P2\ P3\ P4\ P5]$  then  $Nf = [P5\ P4\ P3\ P2\ P1]$ .

### 4.2.3 Operations on Polytopes

Next, we review some standard operations on polytopes.

#### 4.2.3.1 Chebyshev Ball

The *Chebyshev's ball* of a polytope  $\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax \leq b\}$  is the largest hyperball inscribed in  $\mathcal{P}$ , i.e.  $\mathcal{B}_r(x_c) = \{x \in \mathbb{R}^n \mid \|x - x_c\|_2 \leq r\}$ , such that  $\mathcal{B}_r \subseteq \mathcal{P}$ . The center and radius of such a hyperball can be easily found by solving the following LP:

$$\max r \tag{4.2a}$$

$$\text{subj. to } A_i x_c + r \|A_i\|_2 \leq b_i, \tag{4.2b}$$

where  $A_i$  denotes the  $i$ -th row of  $A$ . The polytope  $\mathcal{P}$  is empty if and only if the LP (4.2) is infeasible. If  $r = 0$ , then the polytope is lower dimensional. Note that the center of the Chebyshev Ball is not unique, in general, i.e. there can be multiple solutions (e.g. for rectangles).

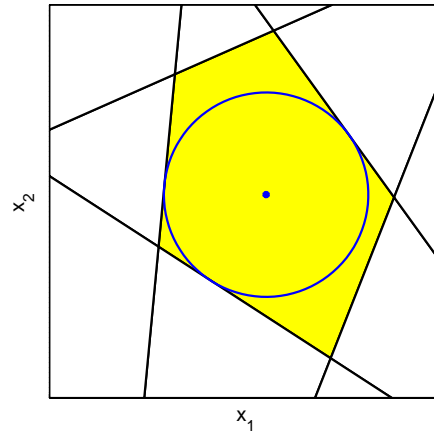
Given a polytope object  $P$ , the center and radius of its Chebyshev's ball can be computed by calling the `chebyball` function:

```
>> [xc, r] = chebyball(P)
```

A sample Chebyshev's ball inscribed in a 2D polytope is shown in Figure 4.2.

#### 4.2.3.2 Affine Transformations

Consider a polytope  $\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax \leq b\}$  and an affine function  $f(x) = Fx + g$  with  $F \in \mathbb{R}^{n \times n}$  and  $g \in \mathbb{R}^n$ ,  $F$  non-singular. Then the *image* of the



**Fig. 4.2** Illustration of the Chebychev's ball.

polytope  $\mathcal{P}$  under the function  $f(x)$  is the polytope  $\mathcal{Q}$

$$\mathcal{Q} = \{Fx + g \mid Ax \leq b\}. \quad (4.3)$$

To see that  $\mathcal{Q}$  is indeed a polytope, define  $y = Fx + g$ . For invertible  $F$  we get  $x = F^{-1}y - F^{-1}g$ . Substituting this expression into  $\{y \mid Ax \leq b\}$  we get

$$\mathcal{Q} = \{y \mid \underbrace{(AF^{-1})}_{\tilde{A}} y \leq \underbrace{(b + AF^{-1}g)}_{\tilde{b}}\}, \quad (4.4)$$

which is again a polytope in the  $\mathcal{H}$ -representation  $\{y \mid \tilde{A}y \leq \tilde{b}\}$ . Note that dimensions of  $x$  and  $y$  are the same since  $F$  is assumed to be a square, non-singular matrix. This (direct) affine transformation of a polytope can be obtained by MPT using its `range` function:

```
>> Q = range(P, F, g)
```

Two special cases of affine transformations are scalar scaling and translation. Scaling a polytope  $\mathcal{P}$  with a scalar  $\alpha \in \mathbb{R}$  is achieved by

```
>> Q = alpha*P
```

Translation of  $\mathcal{P}$  by a vector  $t \in \mathbb{R}^n$  is obtained by

```
>> Q = P + t
```

Applying the linear transformation  $Fx$  to polytope  $\mathcal{P}$  can also be achieved by multiplying the polytope by a (square) matrix  $F$ :

```
>> Q = F*P
```

Needless to say, all these operators can be combined together, i.e.

```
>> Q = alpha*F*P + t
```

The *inverse image* of  $\mathcal{P}$  under the affine map  $Fx + g$  (with  $F$  not necessarily non-singular) is given by

$$\mathcal{Q} = \{x \mid Fx + g \in \mathcal{P}\}. \quad (4.5)$$

If  $\mathcal{P}$  is given in its  $\mathcal{H}$ -representation, then

$$\mathcal{Q} = \{x \mid A(Fx + g) \leq b\} = \{x \mid \underbrace{(AF)}_{\tilde{A}} x \leq \underbrace{(b - Ag)}_{\tilde{b}}\}. \quad (4.6)$$

The command to compute the inverse image in MPT is called `domain`:

```
Q = domain(P, F, g)
```

#### 4.2.3.3 Orthogonal Projection

Given a polytope  $\mathcal{P} = \{x \in \mathbb{R}^n, y \in \mathbb{R}^m \mid A \begin{bmatrix} x \\ y \end{bmatrix} \leq b\} \subset \mathbb{R}^{n+m}$ , the *orthogonal projection* of  $\mathcal{P}$  onto the  $x$ -space is defined as

$$\text{proj}_x(\mathcal{P}) \triangleq \{x \in \mathbb{R}^n \mid \exists y \in \mathbb{R}^m \text{ s.t. } A \begin{bmatrix} x \\ y \end{bmatrix} \leq b\}. \quad (4.7)$$

In MPT, the orthogonal projection of a polytope  $\mathcal{P}$  on some of its coordinates is achieved by calling the `projection` method:

```
>> Q = projection(P, coordinates_to_project_on)
```

As an example, take a unit box in 3D, centered at the origin, and project it on its 2nd and 3rd coordinate:

```
>> P = unitbox(3)
>> Q = projection(P, [2 3])
```

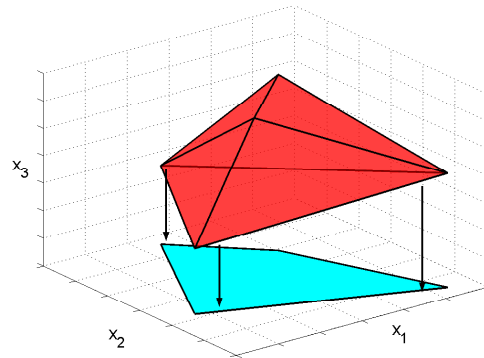
It should be noted that computing projections is considered a hard problem, in general. MPT implements several projection methods and automatically chooses the best depending on the dimension of  $\mathcal{P}$ . An example of projecting a 3D polytope onto a 2D plane is shown in Figure 4.3.

#### 4.2.3.4 Intersection

The *intersection* of two polytopes  $\mathcal{P}_1 = \{x \in \mathbb{R}^n \mid A_1 x \leq b_1\}$  and  $\mathcal{P}_2 = \{x \in \mathbb{R}^n \mid A_2 x \leq b_2\}$  of the same dimension is the polytope

$$\mathcal{Q} = \{x \mid \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} x \leq \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}\}. \quad (4.8)$$

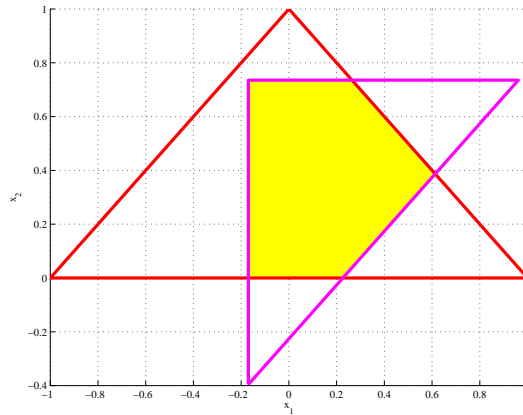
This operation is implemented in MPT using the overloaded `&` operator:



**Fig. 4.3** Illustration of orthogonal projection.

```
>> Q = P1 & P2
```

Note that MPT only computes full-dimensional intersections. Full dimensionality of the result  $Q$  can be checked by `isfulldim(Q)`. An example of the intersection operation is shown in Figure 4.4.



**Fig. 4.4** Full-dimensional intersection of two polytopes.

#### 4.2.3.5 Convex Hull

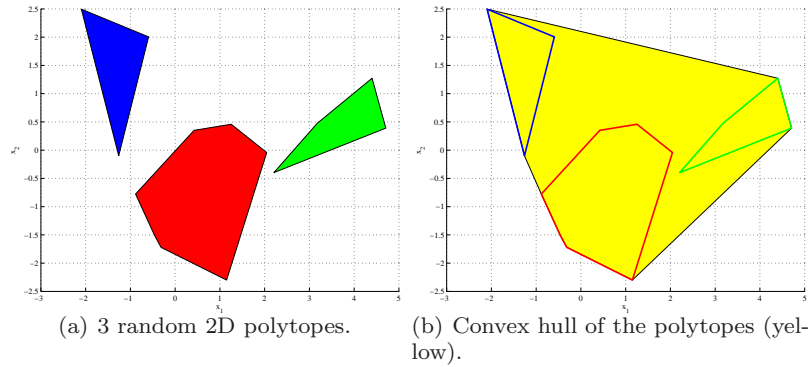
The *convex hull* of two or more polytopes  $\mathcal{P}_i$ ,  $i = 1, \dots, m$  is the smallest convex set which contains them, i.e.:

$$Q = \text{convh}\{\text{vert}(\mathcal{P}_1), \dots, \text{vert}(\mathcal{P}_m)\}, \quad (4.9)$$

where  $\text{vert}(\mathcal{P}_i)$  denotes the extremal vertices of the  $i$ -th polytope. Given an array of polytopes, their convex hull can be computed as follows:

```
>> Q = hull([P1 P2 P3 P4])
```

An example of the convex hull operation is depicted in Figure 4.5.



**Fig. 4.5** Illustration of convex hull.

#### 4.2.3.6 Minkowski Addition

The *Minkowski addition* (also called *set addition*) between two polytopes  $\mathcal{P}_1$  and  $\mathcal{P}_2$  is defined as

$$\mathcal{P}_1 \oplus \mathcal{P}_2 = \{x_1 + x_2 \mid x_1 \in \mathcal{P}_1, x_2 \in \mathcal{P}_2\} \quad (4.10)$$

and is implemented in MPT using the overloaded + (plus) operator:

```
>> Q = P1 + P2
```

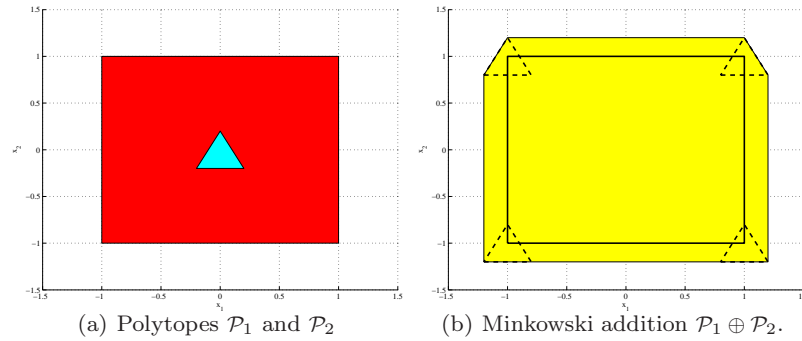
Illustration of Minkowski addition is given in Figure 4.6.

#### 4.2.3.7 Pontryagin Difference

The *Pontryagin difference* (also called *set erosion* or *set dilation*) between two polytopes  $\mathcal{P}_1$  and  $\mathcal{P}_2$  is given by

$$\mathcal{P}_1 \ominus \mathcal{P}_2 = \{x_1 \mid x_1 + x_2 \in \mathcal{P}_1, \forall x_2 \in \mathcal{P}_2\}. \quad (4.11)$$

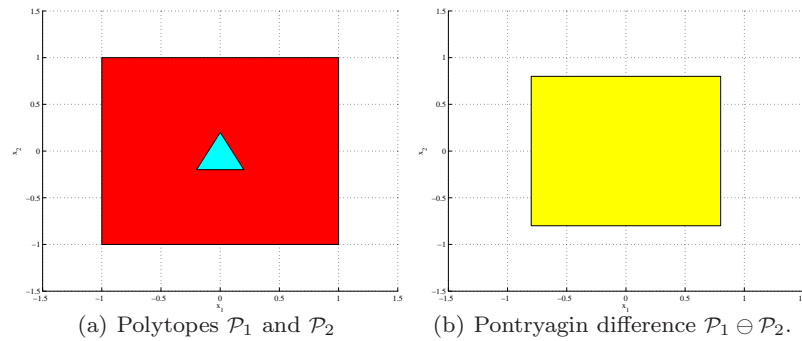
In MPT, this operation is implemented by means of the overloaded - (minus) operator:



**Fig. 4.6** Illustration of Minkowski addition.

```
>> Q = P1 - P2
```

Important to notice is that, in general, the Pontryagin difference and the Minkowski sum operations are not dual to each other, i.e.  $(\mathcal{P}_1 - \mathcal{P}_2) + \mathcal{P}_2 \neq \mathcal{P}_1$ . Graphical interpretation of Pontryagin difference is shown in Figure 4.7.



**Fig. 4.7** Illustration of Pontryagin difference.

#### 4.2.3.8 Set Difference

The *set difference* between two polytopes  $\mathcal{P}_1$  and  $\mathcal{P}_2$  is the set

$$\mathcal{P}_1 \setminus \mathcal{P}_2 = \{x \mid x \in \mathcal{P}_1, x \notin \mathcal{P}_2\}, \quad (4.12)$$

and is implemented by the overloaded `\` (backslash) operator:

```
>> Q = P1 \ P2
```

Note that the set difference of two convex sets can, in general, be non-convex. In such a case the output will be an array of polytopes. An example of the set difference operation is shown in Figure 4.8.

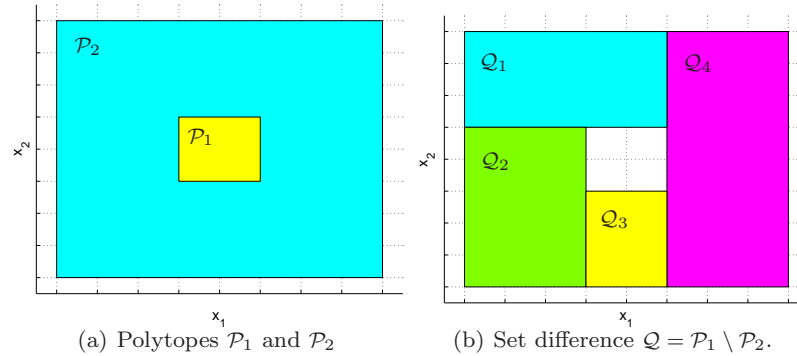


Fig. 4.8 Illustration of the set difference operation.

#### 4.2.4 Functions Overview

A compact overview of the functions and operators described above is given in Tables 4.1 and 4.2. For more information, see the output of `help mpt`, `help polytope` and references therein.

Table 4.1 Computational geometry functions

<code>P=polytope(A,b)</code>	Constructor for the polytope $\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ .
<code>P=polytope(V)</code>	Constructor for creating a $\mathcal{V}$ -represented polytope
<code>double(P)</code>	Access the $\mathcal{H}$ -representation, e.g. <code>[A,b]=double(P)</code> .
<code>extreme(P)</code>	Access the $\mathcal{V}$ -representation, e.g. <code>V=extreme(P)</code> .
<code>display(P)</code>	Displays details about the polytope $\mathcal{P}$ .
<code>nx=dimension(P)</code>	Returns dimension of a given polytope $\mathcal{P}$
<code>nc=nconstr(P)</code>	For a polytope $\mathcal{P} = \{x \mid Ax \leq b\}$ returns number of constraints of the $A$ matrix (i.e. number of rows).
<code>[ , ]</code>	Horizontal concatenation of polytopes into an array, e.g. <code>PA=[P1,P2,P3]</code> .
<code>( )</code>	Subscripting operator for polytope arrays, e.g. <code>PA(i)</code> returns the $i$ -th polytope in $PA$ .
<code>length(PA)</code>	Returns number of elements in a polytope array $\mathcal{P}_A$ .
<code>end</code>	Returns the final element of an array.
<code>[c,r]=chebyball(P)</code>	Center $c$ and radius $r$ of the Chebychev ball inside $\mathcal{P}$ .
<code>bool=isfullldim(P)</code>	Checks if polytope $\mathcal{P}$ is full dimensional.
<code>bool=isinside(P,x)</code>	Checks if $x \in \mathcal{P}$ . Works also for polytope arrays.



**Table 4.2** Operators defined for polytope objects

$P == Q$	Check if two polytopes are equal ( $\mathcal{P} = \mathcal{Q}$ ).
$P \sim= Q$	Check if two polytopes are not-equal ( $\mathcal{P} \neq \mathcal{Q}$ ).
$P >= Q$	Check if $\mathcal{P} \supseteq \mathcal{Q}$ .
$P <= Q$	Check if $\mathcal{P} \subseteq \mathcal{Q}$ .
$P > Q$	Check if $\mathcal{P} \supset \mathcal{Q}$ .
$P < Q$	Check if $\mathcal{P} \subset \mathcal{Q}$ .
$P \& Q$	Intersection of two polytopes, $\mathcal{P} \cap \mathcal{Q}$ .
$P   Q$	Union of two polytopes, $\mathcal{P} \cup \mathcal{Q}$ .
$P + Q$	Minkowski sum, $\mathcal{P} \oplus \mathcal{Q}$ (cf. (4.10)).
$P - Q$	Pontryagin difference, $\mathcal{P} \ominus \mathcal{Q}$ (cf. (4.11)).
$P \setminus Q$	Set difference operator (cf. (4.12)).
$B=\text{bounding\_box}(P)$	Minimal hyper-rectangle containing a polytope $\mathcal{P}$ .
$Q=\text{range}(P, A, f)$	Affine transformation of a polytope. $Q = \{Ax + f \mid x \in \mathcal{P}\}$ .
$Q=\text{domain}(P, A, f)$	Compute polytope that is mapped to $\mathcal{P}$ . $Q = \{x \mid Ax + f \in \mathcal{P}\}$ .
$R=\text{projection}(P, \text{dim})$	Orthogonal projection of $\mathcal{P}$ onto coordinates given in <code>dim</code> (cf. (4.7))

### 4.3 Exercises

**Exercise 4.1.** Perform following tasks:

1. Create a polytope  $P$  as the 2D unit box centered at the origin, i.e.  $P = \{(x_1, x_2)^T \mid -1 \leq x_1 \leq 1, -1 \leq x_2 \leq 1\}$ .
2. Create an another polytope  $Q$  as the convex hull of points  $v_1 = (-1, 0)^T$ ,  $v_2 = (1, 0)^T$ ,  $v_3 = (0, 1)^T$ .
3. Plot  $P$  in red and  $Q$  in blue (don't forget to use `figure` to open a new plot).
4. Is  $P$  a subset of  $Q$ ? Check this using vertices of  $Q$ .
5. Is  $Q$  as subset of  $P$ ? Check this using vertices of  $P$ .

**Exercise 4.2.** MPT offers an easy way to perform affine transformations of polytopes. These include:

- Shifting the polytope  $P$  in the direction given by a vector  $v$  leads a new polytope  $R = \{x + v \mid x \in P\}$ . It can be obtained by:  
`v = [0.8; 1.4]; R = P + v`
- Scaling  $P$  by a scalar factor  $\alpha \neq 0$  gives  $R = \{\alpha x \mid x \in P\}$ :  
`alpha = 0.6; R = alpha*P`
- Affine transformation of  $P$  by an invertible matrix  $M$  gives  $R = \{Mx \mid x \in P\}$ :  
`M = [1.5 0; 0 1]; R = M*P`

Now practice on the triangle defined in Exercise 1:

1. Let  $Q1$  denote the shift of  $Q$  in the direction of the vector  $(0.5, -0.2)^T$
2. Let  $Q2$  be  $Q1$  scaled by a factor of 0.8
3. Let  $Q3$  represent  $Q2$  rotated by  $45^\circ$
4. Plot  $Q$  in red,  $Q1$  in green,  $Q2$  in blue and  $Q3$  in magenta. Moreover, plot all polytopes in the same graph using wireframe.

Hints:

- Rotation is achieved by applying the following matrix transformation with  $\theta$  being the angle in radians:

$$M = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \quad (4.13)$$

- To plot several polytopes in the same graph, do it in a standard MATLAB fashion, i.e. `plot(P1, 'r', P2, 'b', 'P3', 'g')`
- To plot using wireframe, use additional options:

```
opts = struct('wire', 1, 'linewidth', 3);
plot(P1, 'r', P2, 'b', P3, 'g', opts)
```

**Exercise 4.3.** Now concentrate only on  $Q$  and  $Q3$  from the previous exercise. It is obvious from the plot that these two polytopes overlap, i.e. they have a non-empty intersection. To check this, use `dointersect`:

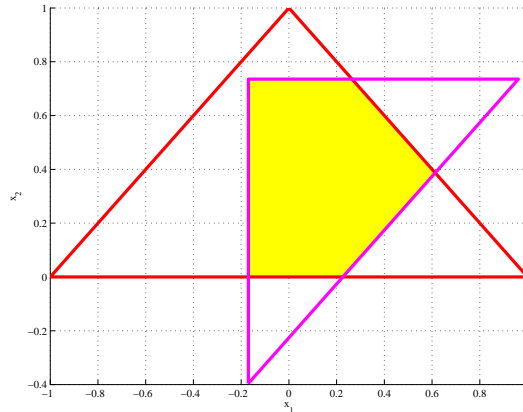
```
dointersect(Q3, Q)
ans =
     1
```

Since the intersection of any two polytopes is again a polytope, the intersection can be computed explicitly by calling the overloaded `&` operator:

```
Q4 = Q3 & Q
```

To visualize the intersection, we plot  $Q4$  in solid yellow and  $Q3$  and  $Q$  using wireframes:

```
% a combined solid and wireframe plot cannot be achieved
% by a single command, therefore we split it in two steps:
opts = struct('wire', 1, 'linewidth', 3);
plot(Q4, 'y');
hold on; plot(Q, 'r', Q3, 'm', opts); hold off
```



**Fig. 4.9** Intersection of polytopes  $Q3$  and  $Q$ .

Now practice: Plot the intersection of  $Q$ ,  $Q2$  and  $Q3$ .

Hint: you can use `&` in a chain, i.e. `Q & Q2 & Q3`.

**Exercise 4.4.** The set difference operation between polytopes  $P$  and  $Q$  is defined as

$P \setminus Q = \{x \mid x \in P, x \notin Q\}$ . In MPT, this is achieved by using the `\` operator:

```
D = Q3 \ Q;
plot(D, 'g')
hold on; plot(Q, 'r', Q3, 'm', opts); hold off
```

Notice that the set difference between two convex sets is not necessarily convex. In fact, the example above illustrates such case and we have  $D$  represented as an *array* of two polytopes:

```
D
D=
```

```
Polytope array: 2 polytopes in 2D
```

The number of elements of  $D$  can be determined by `length(D)` and its components can be accessed using standard MATLAB indexing, e.g.

```
D1 = D(1);
D2 = D(2);
D3 = D(end); % same as D(2)
```

You can also create a polytope array on your own, e.g.

```
polyarray = [D(1) Q D(2) Q3]
```

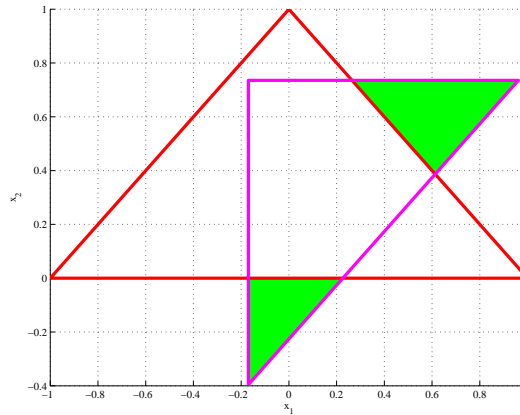
Now practice: Plot the set difference between  $Q$  and  $Q3$  (notice that ordering makes a difference). How many polytopes define the difference?

**Exercise 4.5.** The Minkowski addition operation  $\mathcal{P}_1 \oplus \mathcal{P}_2$ , described in Section 4.2.3.6, is easy to implement if the polytopes  $\mathcal{P}_1$  and  $\mathcal{P}_2$  are given in their  $\mathcal{V}$ -representation, i.e.  $\mathcal{P}_1 = \text{convh}\{x_1, \dots, x_m\}$ ,  $\mathcal{P}_2 = \text{convh}\{y_1, \dots, y_p\}$ . Then

$$\mathcal{P}_1 \oplus \mathcal{P}_2 = \text{convh}\{x_i + y_j, \forall i, \forall j\}. \quad (4.14)$$

Now assume that  $\mathcal{P}_1$  and  $\mathcal{P}_2$  are as follows:

```
>> P1 = unitbox(2)
>> P2 = unitbox(2)*0.1
```



**Fig. 4.10** Set difference between  $Q3$  and  $Q$ .

i.e. hypercubes (in 2D), centered at the origin. The task is to implement Minkowski addition via (4.14). Hint: use the `extreme` function to enumerate vertices of  $\mathcal{P}_1$  and  $\mathcal{P}_2$ . Compare your result to the output of MPT's internal implementation, i.e. to `P1+P2`.

**Exercise 4.6.** Recall that Minkowski addition of two polytopes is given by  $\mathcal{P}_1 \oplus \mathcal{P}_2 = \{x + y \mid x \in \mathcal{P}_1, y \in \mathcal{P}_2\}$ . Assume that the polytopes are given by their  $\mathcal{H}$ -representation, i.e.  $\mathcal{P}_1 = \{x \mid A_1x \leq b_1\}$ ,  $\mathcal{P}_2 = \{y \mid A_2y \leq b_2\}$ . Define a new variable  $z = x + y$ . Then we have

$$\mathcal{P}_1 \oplus \mathcal{P}_2 = \{x + y \mid x \in \mathcal{P}_1, y \in \mathcal{P}_2\} \quad (4.15a)$$

$$= \{x + y \mid A_1x \leq b_1, A_2y \leq b_2\} \quad (4.15b)$$

$$= \{z \mid z = x + y, A_1x \leq b_1, A_2y \leq b_2\} \quad (4.15c)$$

$$= \{z \mid A_1x \leq b_1, A_2(z - x) \leq b_2\} \quad (4.15d)$$

$$= \left\{ z \mid \begin{bmatrix} 0 & A_1 \\ A_2 & -A_2 \end{bmatrix} \begin{bmatrix} z \\ x \end{bmatrix} \leq \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \right\} \quad (4.15e)$$

$$= \{z \mid \tilde{A} \begin{bmatrix} z \\ x \end{bmatrix} \leq \tilde{b}\} \quad (4.15f)$$

$$= \{z \mid \exists x, \text{ s.t. } \tilde{A} \begin{bmatrix} z \\ x \end{bmatrix} \leq \tilde{b}\}. \quad (4.15g)$$

Note that in (4.15d) we have eliminated the variable  $y$  by replacing it with  $z - x$ . The last line (4.15g) states that the Minkowski difference can be in fact computed by projecting the polytope  $\begin{bmatrix} 0 & A_1 \\ A_2 & -A_2 \end{bmatrix} \begin{bmatrix} z \\ x \end{bmatrix} \leq \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$  onto the  $z$ -space, cf. (4.7). Your task is to implement this projection-based procedure and compare its output to MPT's native algorithm, available by running `P1+P2`. Assume the following input data:

```
>> P1 = polytope([-1 -1; -1 1; 1 -1; 1 1])
>> P2 = polytope([-0.1 -0.1; 0.1 -0.1; 0 0.1])
```

**Exercise 4.7.** The Pontryagin difference operation discussed in Section 4.2.3.7 was so far defined under the assumption that  $\mathcal{C}$  and  $\mathcal{B}$  in (4.11) are single polytopes. However, the operation is also well-defined when  $\mathcal{C}$  is a polytope array, i.e.  $\mathcal{C} = \cup_i \mathcal{C}_i$ . In such a case, the difference  $\mathcal{G} = \mathcal{C} \ominus \mathcal{B}$  can be obtained by the following procedure:

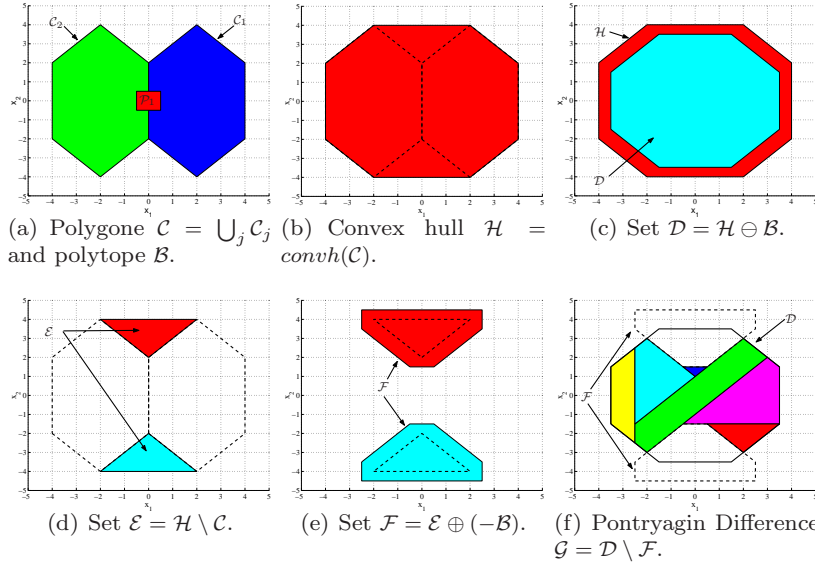
1.  $\mathcal{H} = \text{convh}(\mathcal{C})$
2.  $\mathcal{D} = \mathcal{H} \ominus \mathcal{B}$
3.  $\mathcal{E} = \mathcal{H} \setminus \mathcal{C}$
4.  $\mathcal{F} = \mathcal{E} \oplus (-\mathcal{B})$
5.  $\mathcal{G} = \mathcal{D} \setminus \mathcal{F}$

The algorithm is explained graphically in Figure 4.11. Assume the following input data:

```
>> C1 = polytope([0 -2; 0 2; 2 4; 2 -4; 4 2; 4 -2])
>> C2 = polytope([-4 -2; -4 2; -2 4; -2 -4; 0 -2; 0 2])
```

```
>> C = [C1 C2]
>> B = 0.5*unitbox(2)
```

and implement the procedure above using MPT's basic operators summarized in Table 4.2. Compare the output to the default implementation of C-B.



**Fig. 4.11** Pontryagin difference for non-convex sets.

**Exercise 4.8.** Given an autonomous linear discrete-time system  $x(k+1) = Ax(k)$ , the set of initial conditions  $x(0) \in X_i$  and a target region  $X_f$ , the following question is of imminent importance when investigating safety: will the system states hit the target region in  $N$  steps? We will employ the MPT polytope library to give a yes/no answer to such a question by computing forward reachable sets.

The set of states  $x(k+1)$  which are reachable, in one time step, from  $x(k) \in X_k$  is given by

$$X_{k+1} = \{Ax \mid x \in X_k\} \quad (4.16)$$

When  $X_k$  is an  $\mathcal{H}$ -polytope given by  $X_k = \{x \mid Mx \leq L\}$ ,  $X_{k+1}$  is represented

$$X_{k+1} = \{x \mid MA^{-1}x \leq L\} \quad (4.17)$$

In MPT, given a polytope  $\mathbf{X}k$ , the one-step reachable set  $\mathbf{X}k1$  is easily computed by applying the affine matrix transformation

$$\mathbf{X}k1 = \mathbf{A} * \mathbf{X}k$$

Assume the following input data:

```
A = [0.5, -0.75; 0.75, 0.5];
Xi = unitbox(2) + [3; 0]; % unit box centered at [3; 0]
Xf = 0.5*unitbox(2);      % box with sides of +/- 0.5
                           % centered at [0; 0]
N = 10
```

Now answer the following questions:

1. Will the system states, starting from  $X_i$ , hit the target region in  $N$  steps?
2. If no, by how much would  $N$  need to be increased?
3. Plot the initial set, the terminal region and the reachable sets.

Hints:

- Apply (4.17) in a loop for  $k = 1:N$  starting with  $X_1 = X_i$ .
- Use a growing array to store the intermediate results, e.g.

```
R = polytope;
for k = 1:N
    % compute polytope P
    R = [R P]; % add P at the end of the array
end
```

- Use the `dintersect` function to detect hit of the target (remember Exercise 5).
- You can plot the whole polytope array  $R$  in one shot by `plot(R)`.

**Exercise 4.9.** Given is a discrete-time, LTI system  $x_{k+1} = Ax_k + Bu_k$  subject to constraints  $x_{\min} \leq x_k \leq x_{\max}$ ,  $u_{\min} \leq u_k \leq u_{\max}$ . Take the following data:

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}, \quad x_{\min} = \begin{bmatrix} -5 \\ -5 \end{bmatrix}, \quad x_{\max} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \quad u_{\min} = -1, \quad u_{\max} = 1$$

Let's stabilize the system by an LQR feedback  $u_k = -Kx_k$  with  $K = \text{dlqr}(A, B, \text{eye}(2), 1)$ , which gives  $K = [0.52 \ 0.94]$ . Obviously, such a controller doesn't take constraints into account.

However, there are some states for which all constraints are satisfied, e.g.

```
x = [0.1; 0.1]; % clearly satisfies state constraints
u = -K*x = -0.146; % clearly satisfies input constraints
```

and some for which they are violated, e.g.:

```
x = [1; 1]; % satisfies state constraints
u = -K*x = -1.46 % but violates input constraints
```

Now comes the question: what is the set of states for which all constraints hold? Can it be represented by a polytope?

As you might have guessed, the answer can be found using the MPT's polytope library. In fact, the set of states for which all state and input constraints are satisfied can be represented by

$$\Psi = \{x \mid x_{\min} \leq x_k \leq x_{\max}, u_{\min} \leq u_k \leq u_{\max}\}, \quad (4.18)$$

Using the fact that  $u_k = -Kx_k$  leads to

$$\Psi = \{x \mid x_{\min} \leq x_k \leq x_{\max}, u_{\min} \leq -Kx_k \leq u_{\max}\}, \quad (4.19)$$

which can be rewritten using a compact matrix formulation as

$$\Psi = \left\{x \mid \begin{bmatrix} I \\ -I \end{bmatrix} x \leq \begin{bmatrix} x_{\max} \\ -x_{\min} \end{bmatrix}, \begin{bmatrix} -K \\ K \end{bmatrix} x \leq \begin{bmatrix} u_{\max} \\ -u_{\min} \end{bmatrix} \right\} \quad (4.20)$$

which clearly is an intersection of two polytopes in the  $x$  space, hence a polytope itself.

Can you compute and plot the set  $\Psi$  using the numerical data above? Verify constraint satisfaction by checking the extremal vertices of  $\Psi$ .

**Exercise 4.10.** We still consider the LQR case from the previous example. We have seen that for any state  $x \in \Psi$  the LQR feedback  $u = -Kx$  respects constraints. But for how long? Let's investigate one of the corner points of  $\Psi$ :

```
E = extreme(PSI); x = E(1, :)' ; u = -K*x
u =
    1.0000
```

What is now the successor state  $x_{k+1} = Ax_k + Bu_k$ ?

```
xn = A*x + B*u
xn =
    2.1713
   -3.3287
```

But this point is outside of  $\Psi$ !

```
isinside(PSI, xn)
ans =
    0
```

which means that for this new state some constraints would be violated:

```
u = -K*xn
u =
    2.0002
```



In other words,  $\Psi$  does not possess the *set invariance* property. In simple words, a set  $\Phi_\infty$  is invariant if, for all starting points  $x_0$  inside such set, all future trajectories  $x_k$  remain in the set  $\forall k \geq 0$ . Set invariance is crucial for MPC in order to guarantee stability of the closed-loop system.

Now that we checked that  $\Psi$  is not invariant, can we compute its invariant subset  $\Phi_\infty$ ? Following definition might help:

$$\Phi_\infty = \{x_k \mid x_k \in \Psi, (A - BK)x_k \in \Psi, \forall k \geq 0\} \quad (4.21)$$

Following iterative algorithm can be used to compute  $\Phi_\infty$ :

**Step 1:**  $k = 1, \Psi_k = \Psi$

**Step 2:** repeat

**Step 3:**  $\Psi_{k+1} = \{x \mid x \in \Psi_k, (A - BK)x \in \Psi_k\}$

**Step 4:**  $k = k + 1$

**Step 5:** until  $\Psi_k \neq \Psi_{k-1}$

**Step 6:**  $\Phi_\infty = \Psi_k$

The only non-trivial stage is Step 3. Remember the forward reach sets of Example 4? Take a look again at (4.16) and you should spot some similarities. In fact, the set computed in Step 3 is the *one-step backwards reachable set*, i.e. the subset of  $\Psi_k$  which remains in  $\Psi_k$  for one time step. Since forward reach sets were computed by  $X_{k+1} = AX_k$ , we can compute the backward reach set by  $X_{k+1} = A^{-1}X_k$ , intersected with  $X_k$  itself, i.e.

$$\text{PSI}(k+1) = \text{PSI}(k) \ \& \ \text{inv}(A-B*K)*\text{PSI}(k)$$

The rest of the solution should be clear. You will, however, need to replace **repeat-until** by a **while-end** statement and negate the stopping condition accordingly.

Now it's your turn: compute the invariant subset of PSI and plot it.

Note that many theoretical subtleties were untold in this story. One of them being that the presented algorithm has no guarantee of finite-time convergence, there you should place an upper limit on the number of iterations. 10 is a safe choice.

## 4.4 Solutions

**Solution 4.1 (for Exercise 4.1).**

1. Create a polytope P as the 2D unit box centered at the origin, i.e.  $P = (x_1, x_2)^T \mid -1 \leq x_1 \leq 1, -1 \leq x_2 \leq 1$ .

```

% convert all inequalities into the form [a1 a2]*[x1; x2]
% <= b:
% x_1 <= 1 i.e. [ 1 0]*[x1; x2] <= 1
% -x_1 <= 1 i.e. [-1 0]*[x1; x2] <= 1
% x_2 <= 1 i.e. [ 0 1]*[x1; x2] <= 1
% -x_2 <= 1 i.e. [ 0 -1]*[x1; x2] <= 1
% now stack all inequalities into a matrix:
A = [1 0; -1 0; 0 1; 0 -1];
b = [1; 1; 1; 1];
P = polytope(A, b);

```

2. Create another polytope Q as the convex hull of points  $v_1 = (-1, 0)^T$ ,  $v_2 = (1, 0)^T$ ,  $v_3 = (0, 1)^T$ .

```

v1 = [-0.5; 0]; v2 = [0.5; 0]; v3 = [0; 0.5];
Q = polytope([v1 v2 v3]'); % don't forget the transpose!

```

3. Plot P in red and Q in blue (don't forget to use `figure` to open a new plot).

```

figure; plot(P, 'r')
figure; plot(Q, 'b')

```

4. Is P a subset of Q? Check this using vertices of Q. Or a one-liner...

```

P <= Q
ans =
    0

```

5. Is Q as subset of P? Check this using vertices of P. Or a one-liner...

```

Q <= P
ans =
    1

```

#### Solution 4.2 (for Exercise 4.2).

1. Let Q1 denote the shift of Q in the direction of the vector  $(0.5, -0.2)^T$

```

Q1 = Q + [0.5; -0.2];

```

2. Let Q2 be Q1 scaled by a factor of 0.8

```

Q2 = 0.8*Q1;

```

3. Let Q3 represent Q2 rotated by  $45^\circ$

```

th = 45/180*pi; M = [cos(th) -sin(th); sin(th) cos(th)];
Q3 = M*Q2;

```

4. Plot Q in red, Q1 in green, Q2 in blue and Q3 in magenta. Moreover, plot all polytopes in the same graph using wireframe.

```

opts = struct('wire', 1, 'linewidth', 3);
plot(Q, 'r', Q1, 'g', Q2, 'b', Q3, 'm', opts)
axis equal

```

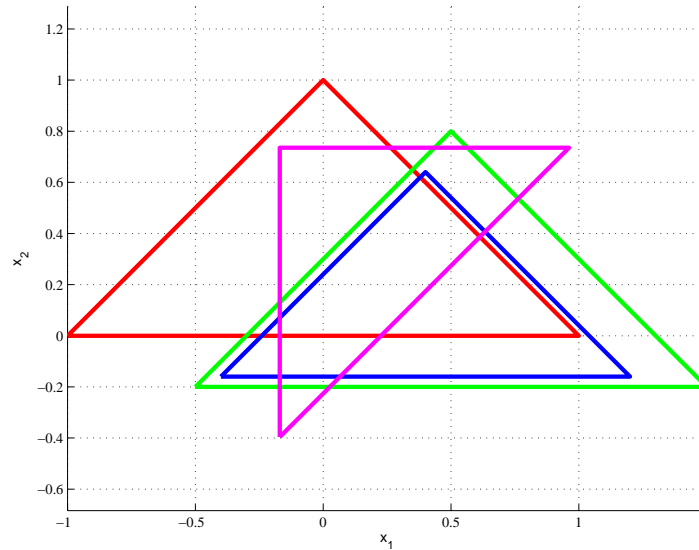


Fig. 4.12 Resulting plot for Exercise 4.2.

**Solution 4.3 (for Exercise 4.3).** Plot the intersection of  $Q$ ,  $Q2$  and  $Q3$ :

```

% quick way:
I = Q & Q2 & Q3

% longer way:
I = Q & Q2;
I = I & Q3;

opts = struct('wire', 1, 'linewidth', 3);
plot(I, 'y');
hold on; plot(Q, 'r', Q1, 'g', Q2, 'b', Q3, 'm', opts); hold
off

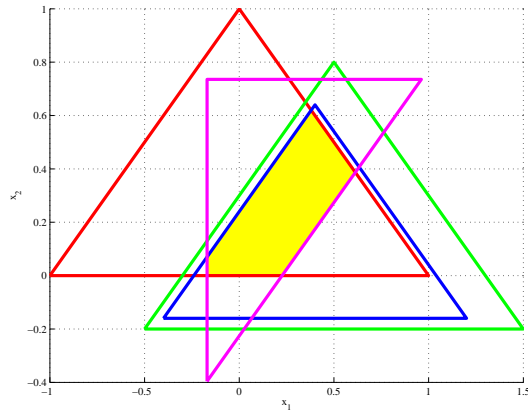
```

**Solution 4.4 (for Exercise 4.4).** Plot the the set difference between  $Q$  and  $Q3$

```

D = Q \ Q3;
plot(D, 'g');
hold on; plot(Q, 'r', Q3, 'm', opts); hold off

```



**Fig. 4.13** Intersection of polytopes Q, Q2, and Q3.

How many polytopes define the difference?

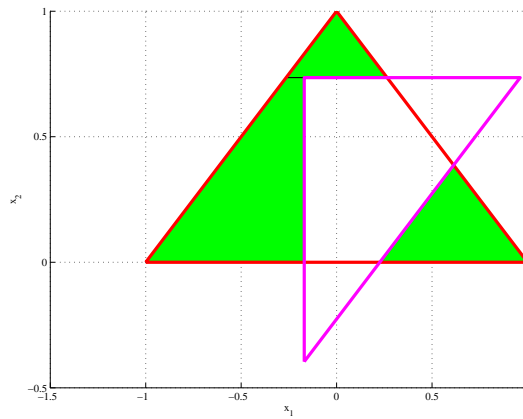
```
length(D)
```

```
ans =
```

```
3
```

**Solution 4.5 (for Exercise 4.5).**

```
% input data
P1 = unitbox(2);
P2 = unitbox(2)*0.1;
```



**Fig. 4.14** Set difference between Q and Q3.

```

% enumerate extremal vertices
V1 = extreme(P1);
V2 = extreme(P2);
nv1 = size(V1, 1); % number of vertices of P1

% initialize the store of vertices
V = [];

% for each vertex of V1, add v1+V2 to the store
for i = 1:size(V1, 1)
    v1 = V1(i, :);
    V = [V; repmat(v1, nv1, 1)+V2];
end

% compute the minkowski addition as convex hull
sum = polytope(V);

% is the output equal to MPT's native implementation?
sum == (P1+P2)

ans =

    1

```

**Solution 4.6 (for Exercise 4.6).** Here is a general solution:

```

P1 = polytope([-1 -1; -1 1; 1 -1; 1 1]);
P2 = polytope([-0.1 -0.1; 0.1 -0.1; 0 0.1]);

[A1, b1] = double(P1);
[A2, b2] = double(P2);
At = [zeros(size(A1, 1), size(A2, 2)), A1; A2, -A2];
bt = [b1; b2];
Pt = polytope(At, bt);
sum = projection(Pt, 1:size(A2, 2));

% is the sum equal to the output of P1+P2?
sum == (P1+P2)

ans =

    1

```

**Solution 4.7 (for Exercise 4.7).** The implementation literally follows the five steps listed in Exercise 4.7:

```

% input data
C1 = polytope([0 -2; 0 2; 2 4; 2 -4; 4 2; 4 -2]);
C2 = polytope([-4 -2; -4 2; -2 4; -2 -4; 0 -2; 0 2]);
C = [C1 C2];
B = 0.5*unitbox(2);

% the algorithm
H = hull(C);
D = H-B;
E = H\C;
F = E+(-B);
G = D\F;

% is the output equal to C-B?
G == (C-B)

ans =

     1

```

**Solution 4.8 (for Exercise 4.8).** Hit or miss in 10 steps?

```

% input data
Xi = unitbox(2) + [3; 0];
Xf = 0.5*unitbox(2);
A = [0.5, -0.75; 0.75, 0.5];
N = 10;

% initialize the data storage
S = Xi;
for k = 1:N
    % compute the set Xk1 = A*Xk
    r = A*S(end);

    % add Xk1 at the end of S
    S = [S r];
end

% plot the sets (remember that Xi is part of S)
plot(Xf, 'k', S)

% hit or miss?
if dointersect(Xf, S)
    disp('Target_hit')
else
    disp('Target_missed')
end

```

end

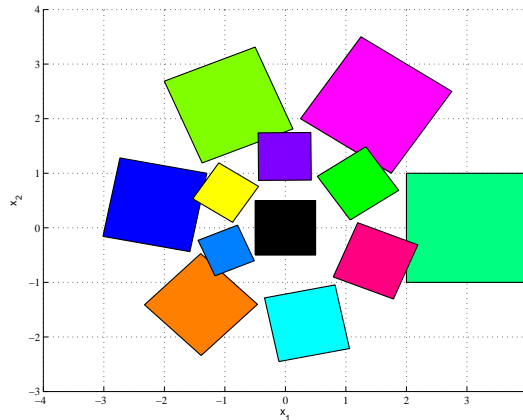


Fig. 4.15 Reachable sets for  $N = 10$  show a miss.

If a miss, how large must  $N$  be for a hit?

```

N = 0;
S = Xi;
while ~dintersect(S, Xf)
    N = N + 1;
    r = A*S(end);
    S = [S r];
end
plot(Xf, 'k', S);
N

```

**Solution 4.9 (for Exercise 4.9).**

```

% input data
A = [1 1; 0 1]; B = [1; 0.5]; K = dlqr(A, B, eye(2), 1);
xmax = [5; 5]; xmin = [-5; -5]; umax = 1; umin = -1;

% set of states which satisfy state and input constraints
X = polytope([eye(2); -eye(2)], [xmax; -xmin]);
U = polytope([-K; K], [umax; -umin]);
PSI = X & U

% plot the set
plot(PSI)

% compute extremal vertices of PSI

```

```

E = extreme(PSI);

% check that input and state constraints hold for each vertex
for i = 1:size(E, 1)
    x = E(i, :)';
    u = -K*x;
    if isinside(unitbox(1), u) && isinside(X, x)
        fprintf('Vertex %d OK\n', i);
    else
        fprintf('Constraints violated for vertex %d!\n', i);
    end
end
end

```

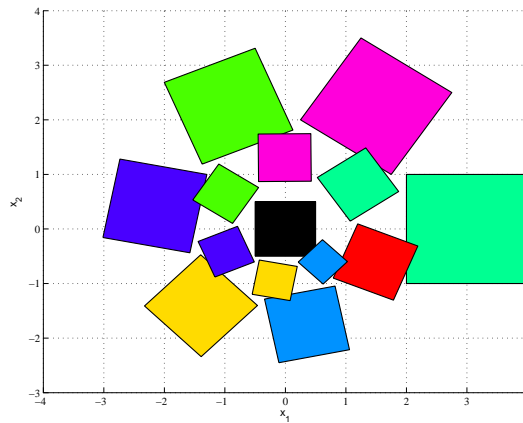


Fig. 4.16 Target hit if  $N \geq 12$ .

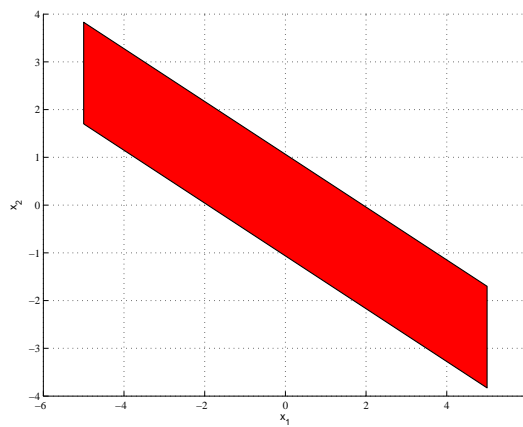


Fig. 4.17 Set of states for which the LQR controller satisfies all constraints.



**Solution 4.10 (for Exercise 4.10).**

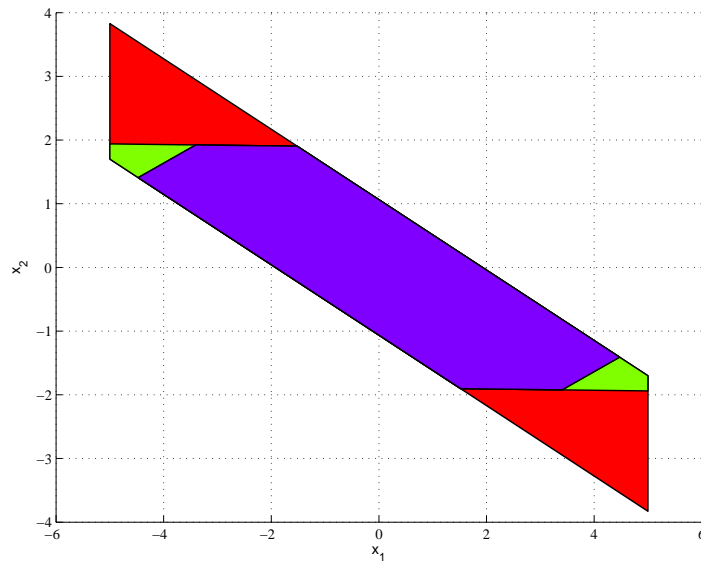
```

% the sets X and U were defined in the previous example
PSI = X & U;
k = 1;
% the algorithm has no guarantee of finite-time convergence
kmax = 10;
while k < kmax
    PSI(k+1) = inv(A-B*K)*PSI(k) & PSI(k);
    k = k + 1;
    if PSI(k) == PSI(k-1), break, end
end
PHI = PSI(end)

% plot all sets
plot(PSI)

% or a one-liner:
PHI = mpt_infset((A-B*K), PSI(1), kmax)

```



**Fig. 4.18** The invariant subset  $\Phi_\infty$  (the innermost region).

## 4.5 Model Predictive Control in MPT

We consider the class of discrete-time, time-invariant systems given by the state-space representation

$$x(t+1) = f(x(t), u(t)), \quad (4.22)$$

where  $x(t)$  is the state vector at time  $t$ ,  $x(t+1)$  is the successor state,  $u(t)$  denotes the control input. Moreover,  $f(\cdot, \cdot)$  is the state-update function which depends on the type of the model, which can either describe a linear time-invariant (LTI) system with

$$f(x(t), u(t)) = Ax(t) + Bu(t), \quad (4.23)$$

or a Piecewise Affine (PWA) system with

$$f(x(t), u(t)) = A_i x(t) + B_i u(t) + f_i \text{ if } \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \in \mathcal{D}_i, \quad (4.24)$$

where the state-update matrices change according to the position of the state-input vector, i.e. the tuple  $(A_i, B_i, f_i)$  is valid if  $\begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \in \mathcal{D}_i$ , where  $\mathcal{D}_i$ ,  $i = 1, \dots, D$  are polytopes of the state-input space. Furthermore, it is assumed that the system variables are subject to constraints

$$x(t) \in \mathcal{X}, \quad u(t) \in \mathcal{U}, \quad \forall t \geq 0, \quad (4.25)$$

where  $\mathcal{X}$  and  $\mathcal{U}$  are polytopic sets.

In MPC, optimal control actions are calculated by formulating and solving a suitable optimization problem, which usually takes the following form:

$$\min_{U_N} \|P_N x_N\|_p + \sum_{k=0}^{N-1} \|Q x_k\|_p + \|R u_k\|_p \quad (4.26a)$$

$$\text{s.t. } x_0 = x(t), \quad (4.26b)$$

$$x_{k+1} = f(x_k, u_k), \quad k = 0, \dots, N-1 \quad (4.26c)$$

$$x_k \in \mathcal{X}, \quad k = 0, \dots, N \quad (4.26d)$$

$$u_k \in \mathcal{U}, \quad k = 0, \dots, N-1 \quad (4.26e)$$

where  $x_k$  and  $u_k$  denote, respectively, the state and input predictions at time instants  $t+k$ , initialized by the measurements of the current state  $x(t)$ , and  $P_N$ ,  $Q$ ,  $R$  are penalty matrices. The prediction is carried out over a finite prediction horizon  $N$ . In addition,  $\|Pz\|_p$  with  $p = 2$  stands for the square of the weighted 2-norm of  $z$ , i.e.  $z^T P z$ . Similarly,  $p = 1$  and  $p = \infty$  denote the 1- and  $\infty$ -norms of some vector.

The aim of MPC is to find the vector  $U_N := [u_0^T, u_1^T, \dots, u_{N-1}^T]^T$  of optimal control inputs which minimizes the cost function (4.26a). If  $f(\cdot, \cdot)$  in (4.26c)

is linear as in (4.23), the MPC formulation can be rewritten as a Quadratic Program (QP) if  $p = 2$ , or as a Linear Program (LP) if  $p \in \{1, \infty\}$ . Finally, if the predictions in (4.26c) are governed by a PWA system, the optimization problem can be stated as a mixed integer QP (MIQP) for  $p = 2$ , or as a mixed integer LP (MILP) for  $p = 1$  or  $p = \infty$ , see e.g. Borrelli (2003).

MPC is usually implemented in the receding horizon (RHMP) fashion. At each sampling instant  $t$ , measurements (or estimates) of  $x(t)$  are obtained first. Then, the optimal control sequence  $U_N(x(t))$  is calculated by assuming  $x(t)$  as the initial condition for problem (4.26). Subsequently, only the first element of the sequence is extracted, i.e.  $u_0(x(t)) = [I, 0, \dots, 0]U_N$  and  $u_0(x(t))$  is fed back to the plant. The whole procedure then repeats at subsequent time instants ad infinitum. This introduces feedback into the scheme.

As explained in the introduction, numerical values of  $U_N$  can be obtained by solving (4.26) as a corresponding LP/QP/MILP/MIQP problem on-line at each sampling time for a particular value of  $x(t)$ . An alternative is to “pre-calculate” the optimal solution to *all possible* values of the initial condition  $x(t)$ :

**Theorem 4.1 (Borrelli (2003)).** *Consider the MPC optimization problem (4.26) with  $f(\cdot, \cdot)$  in (4.26c) being either linear as in (4.23), or piecewise affine as in (4.24). Then the optimizer  $U_N(x(t))$  is a piecewise affine function of  $x(t)$ , i.e.*

$$U_N(x(t)) = F_i x(t) + G_i, \quad \forall x(t) \in \mathcal{R}_i \quad (4.27)$$

where different feedback gains  $F_i, G_i$  switch depending on which region  $\mathcal{R}_i = \{x(t) \mid H_i x(t) \leq K_i\}$ ,  $i = 1, \dots, N_{\text{reg}}$ , contains the state  $x(t)$ .

Theorem 4.1 suggests that the RHMP) feedback  $U_N(x(t))$  can be constructed off-line and stored as a lookup table, which consists of the regions  $\mathcal{R}_i$  and the associated feedback gains  $F_i, G_i$ . The advantage of such an approach is that value of  $U_N(x(t))$  for a particular value of  $x(t)$  can be obtained by simply evaluating the table, as captured by Algorithm 1. The algorithm tra-

---

#### Algorithm 1 Sequential table traversal

---

**INPUT:** Regions  $\mathcal{R}_i$ , feedback laws  $F_i, G_i$ , number of regions  $N_{\text{reg}}$ , state measurement  $x(t)$

**OUTPUT:** Optimal RHMP) control input  $u_0(x(t))$

```

1: for  $r = 1, \dots, N_{\text{reg}}$  do
2:   if  $H_r x(t) \leq K_r$  then
3:      $U_N(x(t)) = F_r x(t) + G_r$ 
4:      $u_0(x(t)) = [I, 0, \dots, 0]U_N(x(t))$ 
5:   return  $u_0(x(t))$ 
6:   end if
7: end for
```

---

verses through the regions sequentially, stopping once it finds a region which

contains  $x(t)$ . In such case the optimal control action is calculated by evaluating the corresponding control law and returned back. Clearly, in the worst case, the algorithm has to search through all regions. But since the operations performed in Steps 2–4 only consist of simple matrix multiplications and additions, for a large class of problems running Algorithm 1 is faster compared to obtaining  $u_0(x(t))$  by solving (4.26) on-line as an LP/QP/MILP/MIQP using off-the-shelf solvers. The second benefit is that the explicit representation of the feedback law allows to further analyze and post-process the MPC controller. For instance, as will be shown in the next section, lookup table-styled controllers allow stability of the closed-loop system to be rigorously analyzed by searching for a certain types of Lyapunov functions.

### 4.5.1 Basic Usage

MPT is a MATLAB-based toolbox which allows for user-friendly design, simulation, analysis, and deployment of MPC-based controllers. Control design is divided into two intuitive steps. At first, the model of the plant is entered either in the LTI form (4.23) or as a PWA system (4.24). We will illustrate the procedure on the double integrator example given by the state-space representation  $x(t+1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x(t) + \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} u(t)$ , which is subject to constraints  $-5 \leq x(t) \leq 5$ ,  $-1 \leq u(t) \leq 1$ . The plant model can then be entered using the command-line interface:

```
sysStruct.A = [1 1; 0 1]; sysStruct.B = [1; 0.5];
sysStruct.C = eye(2); sysStruct.D = zeros(2, 1);
sysStruct.xmin = [-5; -5]; sysStruct.xmax = [5; 5];
sysStruct.umin = -1; sysStruct.umax = 1;
```

With the prediction model at hand, the user then defines parameters of the MPC optimization problem (4.26): the prediction horizon  $N$ , penalty matrices  $P_N$ ,  $Q$ ,  $R$  and the norm  $p$ :

```
probStruct.N = 3;
probStruct.P_N = [1 0; 0 1];
probStruct.Q = [1 0; 0 1];
probStruct.R = 1;
probStruct.norm = 1; % can be 1, 2, or Inf
```

Finally, the MPC controller is synthesized. To solve (4.26) explicitly for all possible initial conditions, one would call

```
>> ctrl = mpt_control(sysStruct,probStruct,'explicit')
```

For the double integrator example and the MPC setup as above, the MPT toolbox constructed the explicit representation of  $U_N$  as a function of  $x(t)$

(cf. (4.27)) in just 0.9 seconds<sup>2</sup>, generating 50 regions. If, instead, the aim is to design an on-line MPC controller, the calling syntax is

```
>> ctrl = mpt_control(sysStruct,probStruct,'online')
```

In both cases the result is stored in the `ctrl` variable, which represents an MPC controller object. The object-oriented philosophy of the Multi-Parametric Toolbox then allows to process the `ctrl` object in a number of ways. The basic one is to ask the controller to provide the optimal value of the control action  $u_0$  for a given initial condition  $x(t)$ . In MPT, the controller object behaves as a function, which allows for the following intuitive syntax:

```
>> x = [-4; 1]; % measured state
>> u = ctrl(x); % evaluation of the controller
```

If the `ctrl` object denotes an explicit MPC controller, the optimal control action `u` is calculated by traversing the lookup table. For the example above, the MPT implementation of Algorithm 1 only takes 2 milliseconds to execute.

If, on the other hand, the object represents an on-line MPC controller, the optimal control input is obtained by solving (4.26) as an optimization problem (in this case an LP) with `x` being its initial condition. MPT toolbox is shipped with several freely available LP/QP/MILP/MIQP solvers to perform this task without further effort from the users. Clearly, as solving the optimization problem is more difficult compared to a simple table lookup, execution of the `u = ctrl(x)` took 50 milliseconds.

### 4.5.2 Closed-loop Simulations

MPT provides several functions to perform closed-loop simulations of MPC-based controllers. The most general approach is to use the specially crafted Simulink block, which allows MPC controllers to be used for virtually any type of simulations. A screenshot of such a block is shown in Figure 4.19 where the controller is connected in a feedback fashion. The Simulink block supports both explicit as well as on-line MPC controllers.

Another option is to use command-line interfaces, which perform closed-loop simulations according to the receding horizon principle:

```
>> [X, U] = sim(ctrl, simmodel, x0, Tsim)
>> [X, U] = simplot(ctrl, simmodel, x0, Tsim)
```

The `sim` function takes a controller object (representing either an explicit or an on-line MPC controller), the simulation model, the initial condition, and the number of simulation steps as the inputs and provides closed-loop evolution of system states and control actions as the output. The `simplot` function behaves in a similar fashion, but, in addition, generates a graphical

<sup>2</sup> 2.4 GHz CPU, 2GB RAM, MATLAB 7.4, MPT 2.6.2

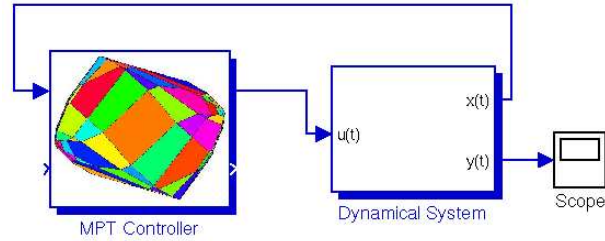


Fig. 4.19 MPT controller block in Simulink.

plot of the closed-loop trajectories as can be seen in Figure 4.20. Note that the simulation model `simmodel` can be different from the prediction model used to design the MPC controller. This allows to verify the controller's performance in situations with model-plant mismatch.

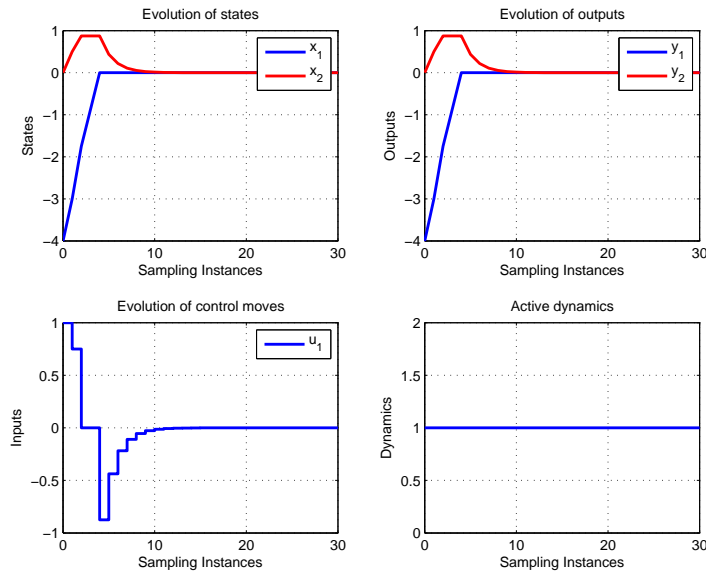


Fig. 4.20 Closed-loop simulation using the `simplot` function.

### 4.5.3 Code Generation and Deployment

As discussed in Section 4.5, if the MPC problem (4.26) is solved off-line and the explicit representation of the feedback law  $U_N(x(t))$  is obtained as a

lookup table, the optimal control action can be obtained at each sampling instant by Algorithm 1. As the algorithm only performs trivial arithmetic operations, it can be implemented using any high- or low-level programming language. Simplicity of the individual operations also allows the table to be traversed using low computational power. This makes the approach of high practical relevance when applying MPC to systems with fast dynamics using low-cost hardware. To push into this direction, the MPT toolbox can automatically generate real-time executable C-code version of Algorithm 1. The export can be initiated by calling

```
>> mpt_exportC(ctrl, 'target_filename')
```

The function generates the files `target_filename.c` and `target_filename.h` which will contain, respectively, the table evaluation code and the table data (the regions  $\mathcal{R}_i$  and the feedback gains  $K_i$  and  $L_i$ , cf. (4.27)). The code can be subsequently linked to any application written in the C language.

Another option is to use the Real-Time Workshop (RTW), which is a de-facto standard code generation tool supporting different types of digital signal processors and other CPUs. MPT provides a RTW-compatible C-code implementation of the Algorithm 1, which makes code generation and deployment a single click operation. To use this option, the controller is first embedded into the Simulink environment using a provided block as shown in Section 4.5.2. The block is subsequently connected to signals from A/D and D/A converters to close the control loop. Then, one simply clicks on the respective icon in the Simulink window to initiate RTW, which compiles the overall control scheme including the lookup table and then automatically downloads it to any supported target CPU.

#### 4.5.4 Advanced MPC using MPT and YALMIP

In the previous section we have shown how to formulate, solve, analyze and deploy MPC controllers using the MPT toolbox. The control design was based on two main input objects – the `sysStruct` structure, which contains information about the system dynamics, and the `probStruct` structure, which defined parameters of the MPC optimization problem (4.26).

Although intuitive and easy to use, the command-line interface outlined in Section 4.5.1 only allows for basic MPC design. In particular, the MPC problem to be formulated and solved can only be adjusted by changing the prediction horizon or the penalty matrices in (4.26a). In this section we describe a new addition to the Multi-Parametric Toolbox which allows more advanced MPC setups to be formulated easily and with low human effort.

The new framework for advanced model predictive control utilizes the YALMIP package, which is a versatile modeling language for construction of generic optimization problems in MATLAB. The design procedure goes as

follows. First, given the model and problem structures, MPT formulates the basic MPC problem (4.26) using YALMIP symbolic decision variables by a single function call:

```
>> [C, obj, V] = mpt_ownmpc(sysStruct, proStruct)
```

which returns constraints (4.26c)–(4.26e) in the variable `C` and the objective function (4.26a) in `obj` formulated using symbolic variables `V.x` and `V.u`. These variables represent the prediction of systems states and inputs through the finite horizon  $N$  stored as cell arrays, i.e. `V.x{k}` denotes  $x_{k-1}$ , and `V.u{k}` stands for  $u_{k-1}$  for some index  $k$ .

Subsequently, the user can modify the MPC problem by adding or custom constraints and/or by changing the objective function. To illustrate this concept, consider again the double integrator example of the previous section. The closed-loop profile of control actions shown in the lower left of Figure 4.20 reveals an abrupt change of the control signal from  $u = 0.8$  to  $u = 0$  at time instant  $t = 2$ . Such a large change can be mitigated, for instance, by adding a constraint which only allows the control signal to change by a certain quantity (say by 0.5) from step to step. This can be achieved by adding the constraints  $-0.5 \leq (u_k - u_{k+1}) \leq 0.5$  to the MPC setup (4.26). In MPT, this is achieved by using a standard YALMIP syntax:

```
>> C = C + [-0.5 <= V.u{k} - V.u{k+1} <= 0.5]
```

with  $k = 1 : N$ . More examples of constraint modification are presented in the sequel.

Once completed, modified constraints and performance objective are passed back to MPT, which converts them into a valid controller object:

```
>> ctrl = mpt_ownmpc(sysStruct, probStruct, C, obj, V, type)
```

If the object should represent an on-line MPC controller, then `type='online'`. If `type='explicit'`, then an explicit representation of the solution to the optimization problem represented by constraints `C` and objective `obj` is pre-computed for all possible initial conditions and stored as a lookup table. In both cases the `ctrl` variable represents a valid controller object, which can be further post-processed or deployed as real-time executable code as illustrated in the previous section.

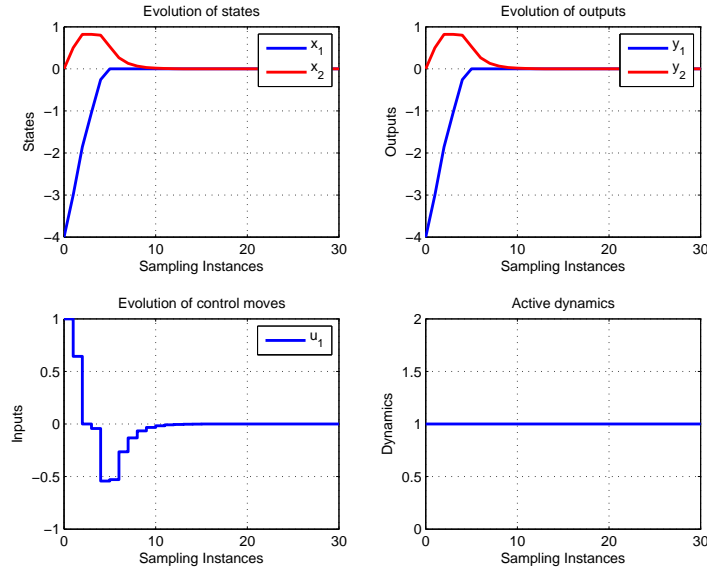
In the sequel we present several examples illustrating how the concept of adding new constraints can be used to achieve typical control tasks.

#### 4.5.4.1 Terminal Constraints

One way of ensuring that the MPC controller will provide closed-loop stability is to add a terminal state constraint  $x_N = 0$ . This requirement can be easily added to the set of constraints by

```
>> C = C + [V.x{end} == 0]
```





**Fig. 4.21** Closed-loop simulation using the `simplot` function.

where  $V.x\{\text{end}\}$  represents the final predicted state  $x_N$ . Another option is to use *terminal set* constraints, i.e. to require that  $x_N \in \mathcal{T}$  for some polytope  $\mathcal{T} = \{x \mid Hx \leq K\}$ . This type of constraints can also be introduced easily by

```
>> C = C + [H*V.x{end} <= K]
```

#### 4.5.4.2 Time-varying Constraints

In the optimization problem (4.26), constraints (4.26d) and (4.26e) are time-invariant. To make these constraints time-varying, one can exploit the fact that the symbolic decision variables  $V.x$  and  $V.u$  are stored as cell arrays indexed by the prediction index  $k$ :

```
>> C = C + [-5 <= V.x{1} <= 5]
>> C = C + [-6 <= V.x{2} <= 6]
>> C = C + [-1 <= V.x{3} <= 1]
```

Such a construction will force the predicted states to stay within of given time-varying bounds.

#### 4.5.4.3 Move Blocking

Move blocking is a popular technique frequently used to reduce complexity of MPC optimization problems by eliminating some degrees of freedom from the sequence  $U_N = [u_0^T, u_1^T, \dots, u_{N-1}^T]^T$ . In practice, usually three types of move blocking are used:

1. Keeping first  $N_c$  moves  $u_0, \dots, u_{N_c}$  free and fixing the remained  $u_{N_c+1} = u_{N_c+2} = \dots = u_{N-1} = 0$ . This can be achieved by

```
>> C = C + [V.u(Nc+1:N-1) == 0]
```

2. First  $N_c$  moves are free, while the remaining control moves are driven by a state feedback  $u = Kx$ , i.e.  $u_{N_c+k} = Kx_k$ ,  $k = 1, \dots, N - N_c - 1$ :

```
for k = Nc+1:N-1
    C = C + [V.u{k} == K*x{k}]
end
```

3. Introduce blocks of control moves which are to stay fixed, e.g.  $u_0 = u_1 = u_2$  and  $u_3 = u_4$ :

```
>> C=C+[V.u{1}==V.u{2}]+[V.u{2}==V.u{3}]
>> C=C+[V.u{4}==V.u{5}]
```

The optimal values of  $u_0$  and  $u_3$  obtained by solving the corresponding optimization problems then pre-determine the values of  $u_1$ ,  $u_2$ , and  $u_4$ .

#### 4.5.4.4 Contraction Constraints

Another way of achieving closed-loop stability is to incorporate a contraction constraint  $\|x_{k+1}\|_p \leq \alpha \|x_k\|_p$ , which requires the controller to push the predicted states towards the origin with some fixed decay rate  $0 \leq \alpha \leq 1$ :

```
for k = 1:N-1
    C=C+[norm(V.x{k+1},1) <= a*norm(V.x{k},1)]
end
```

Notice, however, that norm constraints are nonconvex and the resulting optimization problem will contain binary variables, having negative impact on the time needed to solve the optimization probStruct.

#### 4.5.4.5 Logic Constraints

Constraints involving logic implication and equivalence can be also formulated and automatically translated to a corresponding mixed-integer representation by YALMIP. Take again the double integrator example of the

previous section. In order to make the controller less aggressive, one can require the first optimal control move to satisfy  $-0.5 \leq u_0 \leq 0.5$ , but this constraint should only be imposed if the state is contained in a small box around the origin. If the state is outside of this interval, the control action is to respect  $-1 \leq u_0 \leq 1$ , which is the default constraint already contained in the `sysStruct` structure. To formulate such a control goal with MPT, one can make use of the `implies` operator YALMIP provides:

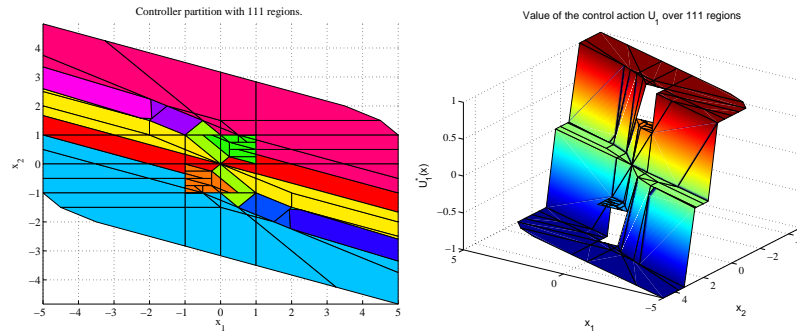
```
>> C = C + [implies(H*V.x{1} <= K, -0.5 <= V.u{1} <= 0.5)]
```

where  $H=[\text{eye}(2); -\text{eye}(2)]$  and  $K=[1;1;1;1]$  represent the area of interest, i.e. the box  $\begin{bmatrix} -1 \\ -1 \end{bmatrix} \leq x(t) \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ . When such a constraint is added to the control setup, an explicit MPC controller can be calculated by `ctrl = mpt_ownmpc(sysStruct, probStruct, C, obj, V)` as a lookup table. Regions of the table and a PWA representation of the optimal feedback law are depicted in Figure 4.22.

Notice that the `implies` operator only works one way, i.e. if  $x(t)$  is in the box delimited by  $Hx(t) \leq K$ , then  $-0.5 \leq u_0 \leq 0.5$  will hold. To restrict the control action to stay within of these limits if and only if  $x(t)$  is in the box, the `iff` operator can be used:

```
>> C = C + [iff(H*V.x{1}<=K,-0.5<=V.u{1}<=0.5)]
```

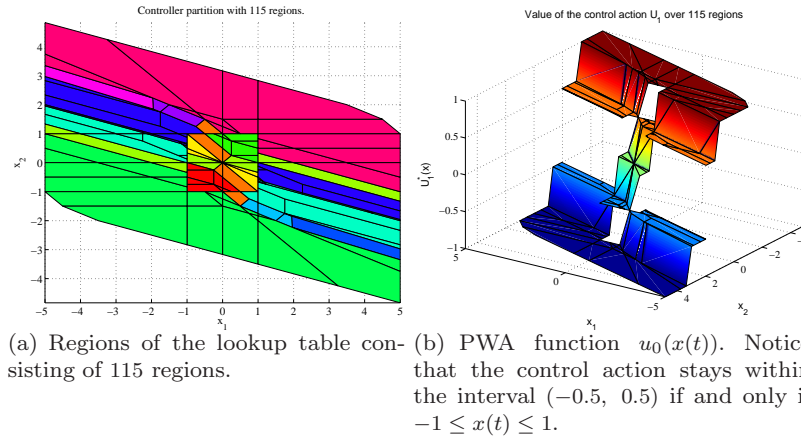
The lookup table corresponding to such a constraint modification is visualized in Figure 4.23.



(a) Regions of the lookup table consisting of 111 regions.

(b) PWA function  $u_0(x(t))$ . Notice that the control action stays within the interval  $(-0.5, 0.5)$  if  $-1 \leq x(t) \leq 1$ .

**Fig. 4.22** Visualization of the lookup table for the control setup with the one-way logic implication constraint  $(-1 \leq x(t) \leq 1) \Rightarrow (-0.5 \leq u_0 \leq 0.5)$ .



**Fig. 4.23** Visualization of the lookup table for the control setup with the login equivalence constraint  $(-1 \leq x(t) \leq 1) \Leftrightarrow (-0.5 \leq u_0 \leq 0.5)$ .

### 4.5.5 Analysis

The toolbox offers broad functionality for analysis of hybrid systems and verification of safety and liveness properties of explicit control laws. In addition, stability of closed-loop systems can be verified using different types of Lyapunov functions.

#### 4.5.5.1 Reachability Analysis

MPT can compute forward  $N$ -steps reachable sets for linear and hybrid systems assuming the system input either belongs to some bounded set of inputs, or when the input is driven by some given explicit control law.

To compute the set of states which are reachable from a given set of initial conditions  $X_0$  in  $N$  steps assuming system input  $u(k) \in \mathcal{U}_0$ , one has to call:

```
R = mpt_reachSets(sysStruct, X0, U0, N);
```

where `sysStruct` is the system structure,  $X_0$  is a polytope which defines the set of initial conditions ( $x(0) \in \mathcal{X}_0$ ),  $U_0$  is a polytope which defines the set of admissible inputs and  $N$  is an integer which specifies for how many steps should the reachable set be computed. The resulting reachable sets  $R$  are returned as a polytope array. We illustrate the computation on the following example:

*Example 4.1.* First we define the dynamical system for which we want to compute reachable sets

```

% define matrices of the state-space object
A = [-1 -4; 4 -1]; B = [1; 1]; C = [1 0]; D = 0;
syst = ss(A, B, C, D);
Ts = 0.02;

% create a system structure by discretizing the continuous-
% time model
sysStruct = mpt_sys(syst, Ts);

% define system constraints
sysStruct.ymax = 10; sysStruct.ymin = -10;
sysStruct.umax = 1; sysStruct.umin = -1;

```

Now we can define a set of initial conditions  $X_0$  and a set of admissible inputs  $U_0$  as polytope objects.

```

% set of initial states
X0 = polytope([0.9 0.1; 0.9 -0.1; 1.1 0.1; 1.1 -0.1]);

% set of admissible inputs
U0 = unitbox(1,0.1); % inputs should be such that |u| <= 0.1

```

Finally we can compute the reachable sets.

```

N = 50;
R = mpt_reachSets(sysStruct, X0, U0, N);

% plot the results
plot(X0, 'r', R, 'g');

```

The reachable sets (green) as well as the set of initial conditions (red) are depicted in Figure 4.24.

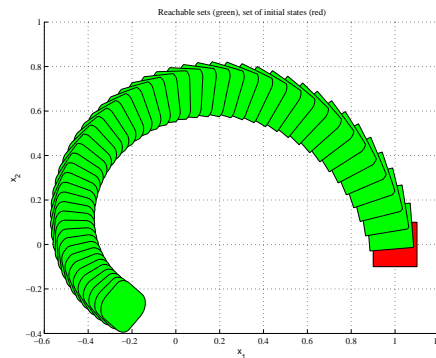


Fig. 4.24 Reachable sets for Example 4.1.

To compute reachable sets for linear or hybrid systems whose inputs are driven by an explicit control law, the following syntax can be used:

```
R = mpt_reachSets(ctrl, X0, N);
```

where `ctrl` is the controller object as generated by `mpt_control`, `X0` is a polytope which defines a set of initial conditions ( $x(0) \in \mathcal{X}_0$ ), and `N` is an integer which specifies for how many steps should the reachable set be computed. The resulting reachable sets `R` are again returned as polytope array.

*Example 4.2.* In this example we illustrate the reachability computation on the *double integrator* example

```
% load system and problem parameters
Double_Integrator

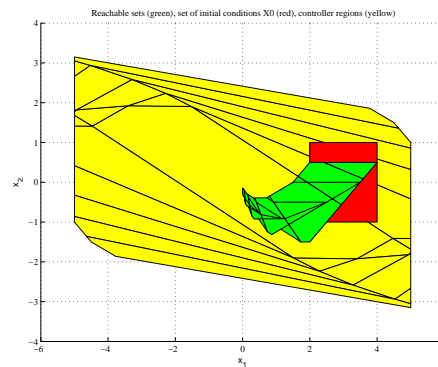
% compute explicit controller
ctrl = mpt_control(sysStruct, probStruct);

% define the set of initial conditions
X0 = unitbox(2,1) + [3;0];

% compute the 5-Steps reachable set
N = 5;
R = mpt_reachSets(ctrl, X0, N);

% plot results
plot(ctrl.Pn, 'y', X0, 'r', R, 'g');
```

The reachable sets (green) as well as the set of initial conditions (red) are depicted on top of the controller regions (yellow) in Figure 4.25.



**Fig. 4.25** Reachable sets for Example 4.2.

#### 4.5.5.2 Verification

Reachability computation can be directly extended to answer the following question: *Do the states of a dynamical system (whose inputs either belong to some set of admissible inputs, or whose inputs are driven by an explicit control law) enter some set of “unsafe” states in a given number of steps?*

*Example 4.3.* In this example we show how to answer the verification question for the first case, i.e. system inputs belong to some set of admissible inputs ( $u(k) \in \mathcal{U}_0$ ). Although we use a linear system here, exactly the same procedure applies to hybrid systems in PWA representation as well.

```
% define matrices of the state-space object
A = [-1 -4; 4 -1]; B = [1; 1]; C = [1 0]; D = 0;
syst = ss(A, B, C, D);
Ts = 0.02;

% create a system structure by discretizing the continuous-
    time model
sysStruct = mpt_sys(syst, Ts);

% define system constraints
sysStruct.ymax = 10; sysStruct.ymin = -10;
sysStruct.umax = 1; sysStruct.umin = -1;

% define the set of initial conditions as a polytope object
X0 = polytope([0.9 0.1; 0.9 -0.1; 1.1 0.1; 1.1 -0.1]);

% set of admissible inputs as a polytope object
U0 = unitbox(1,0.1); % inputs should be such that |u| <= 0.1

% set of final states (the ‘‘unsafe’’ states)
Xf = unitbox(2,0.1) + [-0.2; -0.2];

% number of steps
N = 50;

% perform verification
[canreach, Nf] = mpt_verify(sysStruct, X0, Xf, N, U0);
```

If the system states can reach the set  $X_f$ , `canreach` will be *true*, otherwise the function will return *false*. In case  $X_f$  can be reached, the optional second output argument `Nf` will return the number of steps in which  $X_f$  can be reached from  $X_0$ .

*Example 4.4.* It is also possible to answer the verification question if the system inputs are driven by an explicit control law:

```

% load dynamical system
Double_Integrator

% compute explicit controller
expc = mpt_control(sysStruct, probStruct);

% define set of initial condintions as a polytope object
X0 = unitbox(2,1) + [3;0];

% set of final states (the ‘‘unsafe’’ states)
Xf = unitbox(2,0.1) + [-0.2; -0.2];

% number of steps
N = 10;

% perform verification
[canreach, Nf] = mpt_verify(expc, X0, Xf1, N);

```

#### 4.5.5.3 Invariant Set Computation

For controllers for which no feasibility guarantee can be given a priori, the function `mpt_invariantSet` can compute an invariant subset of a controller, such that constraints satisfaction is guaranteed for all time.

```
ctrl_inv = mpt_invariantSet(ctrl)
```

#### 4.5.5.4 Lyapunov-type Stability Analysis

In terms of stability analysis, MPT offers functions which aim at identifying quadratic, sum-of-squares, piecewise quadratic, piecewise affine or piecewise polynomial Lyapunov functions. If such a function is found, it can be used to show stability of the closed-loop systems even in cases where no such guarantee can be given a priori based on the design procedure. To compute a Lyapunov function, one has to call

```
ctrl_lyap = mpt_lyapunov(ctrl, lyaptype)
```

where `ctrl` is an explicit controller and `lyaptype` is a string parameter which defines the type of a Lyapunov function to compute. Allowed values of the second parameter are summarized in Table 4.3. Parameters of the Lyapunov function, if one exists, will be stored in

```
lyapfunction = ctrl_lyap.details.lyapunov
```

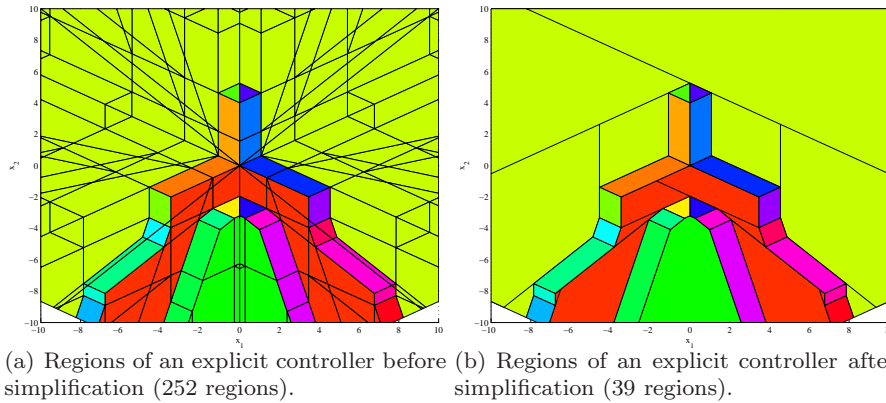


lyaptype	Type of Lyapunov function
'quad'	Common quadratic Lyapunov function
'sos'	Common sum-of-squares Lyapunov function
'pwa'	Piecewise affine Lyapunov function
'pwq'	Piecewise quadratic Lyapunov function
'pwp'	Piecewise polynomial Lyapunov function

**Table 4.3** Allowed values of the `functiontype` parameter in `mpt_lyapunov`.

#### 4.5.5.5 Complexity Reduction

MPT also addresses the issue of complexity reduction of the resulting explicit control laws. As explained in more detail in Section 4.5, the on-line evaluation of explicit control laws involves checking which region of the controller contains a given measured state. Although such an effort is usually small, it can become prohibitive for complex controllers with several thousands or even more regions. Therefore MPT allows to reduce this complexity by simplifying the controller partitions over which the control law is defined. This simplification is performed by merging regions which contain the same expression of the control law. By doing so, the number of regions may be greatly reduced, while maintaining the same performance as the original controller. The results of the merging procedure for a sample explicit controller of a hybrid system is depicted in Figure 4.26.



**Fig. 4.26** Region merging results.

To simplify the representation of a given explicit controller by merging regions which contain the same control law, one has to call:

```
ctrl_simple = mpt_simplify(ctrl)
```

If the function is called as indicated above, a heuristic merging will be used. It is also possible to use optimal merging based on boolean minimization:

```
ctrl_simple = mpt_simplify(ctrl, 'optimal')
```

Note, however, that the optimal merging can be prohibitive for dimensions above 2 due to an exponential complexity of the merging procedure [Geyer \(2005\)](#). See clipping method ([Kvasnica et al, 2011a](#)) in Textbook and approach based on separation functions ([Kvasnica et al, 2011b](#)) in Preprints for alternative ways with better convergence properties and larger reduction ratio.

### 4.5.6 System Structure sysStruct

LTI dynamics can be captured by the following linear relations:

$$x(k+1) = Ax(k) + Bu(k) \quad (4.28a)$$

$$y(k) = Cx(k) + Du(k) \quad (4.28b)$$

where  $x(k) \in \mathbb{R}^{n_x}$  is the state vector at time instance  $k$ ,  $x(k+1)$  denotes the state vector at time  $k+1$ ,  $u(k) \in \mathbb{R}^{n_u}$  and  $y(k) \in \mathbb{R}^{n_y}$  are values of the control input and system output, respectively.  $A$ ,  $B$ ,  $C$  and  $D$  are matrices of appropriate dimensions, i.e.  $A$  is a  $n_x \times n_x$  matrix, dimension of  $B$  is  $n_x \times n_u$ ,  $C$  is a  $n_y \times n_x$  and  $D$  a  $n_y \times n_u$  matrix.

Dynamical matrices are stored in the following fields of the system structure:

```
sysStruct.A = A
sysStruct.B = B
sysStruct.C = C
sysStruct.D = D
```

*Example 4.5.* Assume a double integrator dynamics sampled at 1 second:

$$x(k+1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} u(k) \quad (4.29a)$$

$$y(k) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u(k) \quad (4.29b)$$

In MPT, the above described system can be defined as follows:

```

sysStruct.A = [1 1; 0 1];
sysStruct.B = [1; 0.5];
sysStruct.C = [1 0; 0 1];
sysStruct.D = [0; 0]

```

#### 4.5.6.1 Import of Models from External Sources

MPT can design control laws for discrete-time constrained linear, switched linear and hybrid systems. Hybrid systems can be described in Piecewise-Affine (PWA) or Mixed Logical Dynamical (MLD) representations and an efficient algorithm is provided to switch from one representation to the other form and vice-versa. To increase user's comfort, models of dynamical systems can be imported from various sources:

- Models of hybrid systems generated by the HYSDEL language
- MLD structures generated by the function `mpt_pwa2mld`
- Nonlinear models defined by `mpt_nonlinfcn` template
- State-space and transfer function objects of the Control toolbox,
- System identification toolbox objects
- MPC toolbox objects

In order to import a dynamical system, one has to call

```
sysStruct = mpt_sys(object, Ts)
```

where `object` can be either a string (in which case the model is imported from a corresponding HYSDEL source files), or it can be a variable of one of the above mentioned object types. The second input parameter `Ts` denotes sampling time and can be omitted, in which case `Ts = 1` is assumed.

*Example 4.6.* The following code will first define a continuous-time state-space object which is then imported to MPT:

```

% sampling time
Ts = 1;

% continuous-time model as state-space object
di = ss([1 1; 0 1], [1; 0.5], [1 0; 0 1], [0; 0]);

% import the model and discretize it
sysStruct = mpt_sys(di, Ts);

```

*Note 4.1.* If the state-space object is already in discrete-time domain, it is not necessary to provide the sampling time parameter `Ts` to `mpt_sys`. After importing a model using `mpt_sys` it is still necessary to define system constraints as described previously.

MPT allows to define following types of constraints:

- Min/Max constraints on system outputs
- Min/Max constraints on system states
- Min/Max constraints on manipulated variables
- Min/Max constraints on slew rate of manipulated variables

#### 4.5.6.2 Constraints on System Outputs

Output equation is in general driven by the following relation for PWA systems

$$y(k) = C_i x(k) + D_i u(k) + g_i \quad (4.30)$$

and by

$$y(k) = Cx(k) + Du(k) \quad (4.31)$$

for LTI systems. It is therefore clear that by choice of  $C = I$  one can use these constraints to restrict system states as well. Min/Max output constraints have to be given in the following fields of the system structure:

```
sysStruct.outmax = outmax
sysStruct.outmin = outmin
```

where `outmax` and `outmin` are  $n_y \times 1$  vectors.

#### 4.5.6.3 Constraints on System States

Constraints on system states are optional and can be defined by

```
sysStruct.xmax = xmax
sysStruct.xmin = xmin
```

where `xmax` and `xmin` are  $n_x \times 1$  vectors.

#### 4.5.6.4 Constraints on Manipulated Variables

Goal of each control technique is to design a controller which chooses a proper value of the manipulated variable in order to achieve the given goal (usually to guarantee stability, but other aspects like optimality may also be considered at this point). In most real plants values of manipulated variables are restricted and these constraints have to be taken into account in controller design procedure. These limitations are usually saturation constraints and can be captured by min / max bounds. In MPT, constraints on control input are given in:

```
sysStruct.ymax = inymax
sysStruct.ymin = inymin
```

where `inpmax` and `inpmin` are  $n_u \times 1$  vectors.

#### 4.5.6.5 Constraints on Slew Rate of Manipulated Variables

Another important type of constraints are rate constraints. These limitations restrict the variation of two consecutive control inputs ( $\delta u = u(k) - u(k-1)$ ) to be within of prescribed bounds. One can use slew rate constraints when a “smooth” control action is required, e.g. when controlling a gas pedal in a car to prevent the car from jumping due to sudden changes of the controller action. Min/max bounds on slew rate can be given in:

```
sysStruct.dumax = slewmax
sysStruct.dumin = slewmin
```

where `slewmax` and `slewmin` are  $n_u \times 1$  vectors.

*Note 4.2.* This is an optional argument and does not have to be defined. If it is not given, bounds are assumed to be  $\pm\infty$ .

#### 4.5.6.6 Systems with Discrete Valued Inputs

MPT allows to define system with discrete-valued control inputs. This is especially important in a framework of hybrid systems where control inputs are often required to belong to certain set of values. We distinguish two cases:

1. All inputs are discrete
2. Some inputs are discrete, the rest are continuous

#### 4.5.6.7 Purely Discrete Inputs

Typical application of discrete-valued inputs are various on/off switches, gears, selectors, etc. All these can be modelled in MPT and taken into account in controller design. Defining discrete inputs is fairly easy, all that needs to be done is to fill out

```
sysStruct.Uset = Uset
```

where `Uset` is a cell array which defines all possible values for every control input. If the system has, for instance, 2 control inputs and the first one is just an on/off switch (i.e.  $u_1 = \{0, 1\}$ ) and the second one can take values from set  $\{-5, 0, 5\}$ , it can be defined as follows:

```
sysStruct.Uset{1} = [0, 1]
sysStruct.Uset{2} = [-5, 0, 5]
```

where the first line corresponds to  $u_1$  and the second to  $u_2$ . If the system to be controlled has only one manipulated variable, the cell operator can be omitted, i.e. one could write:

```
sysStruct.Uset = [0, 1]
```

The set of inputs doesn't have to be ordered.

#### 4.5.6.8 Mixed Inputs

Mixed discrete and continuous inputs can be modelled by appropriate choice of `sysStruct.Uset`. For each continuous input it is necessary to set the corresponding entry to `[-Inf Inf]`, indicating to MPT that this particular input variable should be treated as a continuous input. For a system with two manipulated variables, where the first one takes values from a set  $\{-2.5, 0, 3.5\}$  and the second one is continuous, one would set:

```
sysStruct.Uset{1} = [-2.5, 0, 3.5]
sysStruct.Uset{2} = [-Inf Inf]
```

#### 4.5.6.9 Text Labels

State, input and output variables can be assigned a text label which overrides the default axis labels in trajectory and partition plotting ( $x_i$ ,  $u_i$  and  $y_i$ , respectively). To assign a text label, set the following fields of the system structure, e.g. as follows:

```
sysStruct.xlabels = {'position', 'speed'};
sysStruct.ulabels = 'force';
sysStruct.ylabels = {'position', 'speed'};
```

Each field is an array of strings corresponding to a given variable. If the user does not define any (or some) labels, they will be replaced by default strings ( $x_i$ ,  $u_i$  and  $y_i$ ). The strings are used once polyhedral partition of the explicit controller, or closed-loop (open-loop) trajectories are visualized.

### 4.5.7 Problem Structure `probStruct`

Problem structure `probStruct` is a structure which states an optimization problem to be solved by MPT.

#### 4.5.7.1 Quadratic Cost Problems

In case of a performance index based on quadratic forms, the optimal control problem takes the following form:

$$\begin{aligned} \min_{u(0), \dots, u(N-1)} \quad & x(N)^T P_N x(N) + \sum_{k=0}^{N-1} u(k)^T R u(k) + x(k)^T Q x(k) \\ \text{subj. to} \quad & \\ & x(k+1) = f_{dyn}(x(k), u(k), w(k)) \\ & u_{min} \leq u(k) \leq u_{max} \\ & \Delta u_{min} \leq u(k) - u(k-1) \leq \Delta u_{max} \\ & y_{min} \leq g_{dyn}(x(k), u(k)) \leq y_{max} \\ & x(N) \in T_{set} \end{aligned}$$

If the problem is formulated for a fixed prediction horizon  $N$ , we refer to it as to Constrained Finite Time Optimal Control (CFTOC) problem. If  $N$  is infinity, the Constrained Infinite Time Optimal Control (CITOC) problem is formulated. Objective of the optimization is to choose the manipulated variables such that the performance index is minimized.

#### 4.5.7.2 One and Infinity Norm Problems

The optimal control problem with a linear performance index is given by:

$$\begin{aligned} \min_{u(0), \dots, u(N-1)} \quad & \|P_N x(N)\|_p + \sum_{k=0}^{N-1} \|R u(k)\|_p + \|Q x(k)\|_p \\ \text{subj. to} \quad & \\ & x(k+1) = f_{dyn}(x(k), u(k), w(k)) \\ & u_{min} \leq u(k) \leq u_{max} \\ & \Delta u_{min} \leq u(k) - u(k-1) \leq \Delta u_{max} \\ & y_{min} \leq g_{dyn}(x(k), u(k)) \leq y_{max} \\ & x(N) \in T_{set} \end{aligned}$$

where:

- $u$  vector of optimized control inputs
- $N$  prediction horizon
- $p$  norm indicator (can be 1, 2, or `Inf`)
- $Q$  weighting matrix on the states
- $R$  weighting matrix on the manipulated variables
- $P_N$  weight imposed on the terminal state
- $u_{min}, u_{max}$  constraints on the manipulated variable(s)
- $\Delta u_{min}, du_{max}$  constraints on slew rate of the manipulated variable(s)
- $y_{min}, y_{max}$  constraints on the system outputs
- $T_{set}$  terminal set
- the function  $f_{dyn}(x(k), u(k), w(k))$  is the state-update function and is different for LTI and for PWA systems (see Section 4.5.6 for more details).

#### 4.5.7.3 Mandatory Fields

In order to specify which problem the user wants to solve, mandatory fields of the problem structure `probStruct` are listed in Table 4.4.

<code>probStruct.N</code>	prediction horizon
<code>probStruct.Q</code>	weights on the states
<code>probStruct.R</code>	weights on the inputs
<code>probStruct.norm</code>	1, 2 or <code>Inf</code> norms in the cost
<code>probStruct.subopt_lev</code>	level of optimality

Table 4.4 Mandatory fields of the problem structure `probStruct`.

#### 4.5.7.4 Level of Optimality

MPT can handle different setups of control problems. Specifically:

1. The cost-optimal solution that leads a control law which minimizes a given performance index. This strategy is enforced by

$$\text{probStruct.subopt\_lev} = 0$$

The cost optimal solution for PWA systems is currently supported only for linear performance index, i.e. `probStruct.norm = 1` or `probStruct.norm = Inf`.

2. Another possibility is to use the minimum-time setup, i.e. the control law will push a given state to an invariant set around the origin as fast as possible. This strategy usually leads to simpler control laws, i.e. fewer controller regions are generated. This approach is enforced by

$$\text{probStruct.subopt\_lev} = 1$$



3. The last option is to use a low-complexity control scheme. This approach aims at constructing a one-step solution and subsequently a PWQ or PWA Lyapunov function computation is performed to verify stability properties. The approach generally results in a small number of regions and asymptotic stability as well as closed-loop constraint satisfaction is guaranteed. If one wants to use this kind of solution, he/she should set:

```
probStruct.subopt_lev = 2
```

#### 4.5.7.5 Optional Fields

Optional fields are summarized next.

**probStruct.Qy**: used for output regulation. If provided the additional term  $\|Q(y - y_{ref})\|_p$  is introduced in the cost function and the controller will regulate the output(s) to the given references (usually zero, or provided by **probStruct.yref**).

**probStruct.tracking**: **0** – no tracking, resulting controller is a state regulator which drives all system states (or outputs, if **probStruct.Qy** is given) towards origin. **1** – tracking with  $\Delta u$ -formulation. The controller will drive the system states (or outputs, if **probStruct.Qy** is given) to a given reference. The optimization is performed over the difference of manipulated variables ( $u(k) - u(k - 1)$ ), which involves an extension of the state vector by  $nu$  additional states where  $nu$  is the number of system inputs. **2** – tracking without  $\Delta u$ -formulation. The same as **probStruct.tracking=1** with the exception that the optimization is performed over  $u(k)$ , i.e. no  $\Delta u$ -formulation is used and no state vector extension is needed. Note, however, that offset-free tracking cannot be guaranteed with this setting. Default setting is **probStruct.tracking = 0**.

**probStruct.yref**: instead of driving a state to zero, it is possible to reformulate the control problem and rather force the output to zero. To ensure this task, define **probStruct.Qy** which penalizes the difference of the actual output and the given reference.

**probStruct.P\_N**: weight on the terminal state. If not specified, it is assumed to be zero for quadratic cost objectives, or  $P_N = Q$  for linear cost.

**probStruct.Nc**: the control horizon. Specifies the number of free control moves in the optimization problem.

**probStruct.Tset**: a polytope object describing the terminal set. If not provided and **probStruct.norm = 2**, the invariant LQR set around the origin will be computed automatically to guarantee stability properties.

Since MPT 2.6 it is possible to denote certain constraints as soft. This means that the respective constraint can be violated, but such a violation is penalized. To soften certain constraints, it is necessary to define the penalty on violation of such constraints:

- `probStruct.Sx` - if given as a "nx" x "nx" matrix, all state constraints will be treated as soft constraints, and violation will be penalized by the value of this field.
- `probStruct.Su` - if given as a "nu" x "nu" matrix, all input constraints will be treated as soft constraints, and violation will be penalized by the value of this field.
- `probStruct.Sy` - if given as a "ny" x "ny" matrix, all output constraints will be treated as soft constraints, and violation will be penalized by the value of this field.

In addition, one can also specify the maximum value by which a given constraint can be exceeded:

- `probStruct.sxmax` - must be given as a "nx" x 1 vector, where each element defines the maximum admissible violation of each state constraints.
- `probStruct.sumax` - must be given as a "nu" x 1 vector, where each element defines the maximum admissible violation of each input constraints.
- `probStruct.symax` - must be given as a "ny" x 1 vector, where each element defines the maximum admissible violation of each output constraints.

The aforementioned fields also allow to specify that only a subset of state, input, or output constraint should be treated as soft constraints, while the rest of them remain hard. Say, for instance, that we have a system with 2 states and we want to soften only the second state constraint. Then we would write:

```
probStruct.Sx = diag([1 1000])
probStruct.sxmax = [0; 10]
```

Here `probStruct.sxmax(1)=0` tells MPT that the first constraint should be treated as a hard constraint, while we are allowed to exceed the second constraints by at most 10 and every such violation will be penalized by the factor of 1000.

## 4.6 Exercises

**Exercise 4.11.** Consider a model of two liquid tanks given by the following discrete-time state-space model:

$$x(t+1) = \begin{bmatrix} -0.0315 & 0 \\ 0.0315 & -0.0351 \end{bmatrix} x(t) + \begin{bmatrix} 0.0769 \\ 0 \end{bmatrix} u(k)$$

$$y(t) = [0 \ 1] x(t)$$

where the state vector  $x(t) = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}$  represents the deviations of liquid levels from some steady-state levels and  $u(t)$  is the deviation of the liquid inflow from the steady-state value. The state, input, and output variables are assumed to be constrained as follows:

- $\begin{bmatrix} -21 \\ -21 \end{bmatrix} \leq x(t) \leq \begin{bmatrix} 3.5 \\ 3.5 \end{bmatrix}$
- $-17 \leq u(t) \leq 3$
- $-21 \leq y(t) \leq 3.5$

Perform following tasks:

1. Create a `sysStruct` representation of such a system by filling out the structure fields as described in Section 4.5.6.
2. Assign symbolic labels `level 1`, `level 2` to the two state variables, `inflow` to the input, and `level 2` to the output variable.
3. Verify that `sysStruct` contains valid entries by running

```
>> mpt_verifySysStruct(sysStruct);
```

If `sysStruct` is mis-configured, the verification function will output an error.

4. Simulate the evolution of the system for 5 steps, starting from the initial condition  $x(0) = \begin{bmatrix} -5 \\ -10 \end{bmatrix}$  and using the control inputs  $u(t) = -2$ ,  $t = 1, \dots, 10$ . (hint: use the `mpt_simSys` function, see `help mpt_simSys` for more details).

**Exercise 4.12.** For the two-tanks system defined in the previous exercise we would like to synthesize an MPC controller which would minimize the following performance objective:

$$\min \sum_{k=0}^N x_k^T Q x_k + u_k^T R u_k. \quad (4.32)$$

Assume  $N = 6$ ,  $Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ , and  $R = 1$ . Create a problem structure `probStruct` which will reflect this settings (do not forget to set `probStruct.norm=2` to indicate that a quadratic performance objective should be used). Verify consistency of the generated structure by running

```
>> mpt_verifyProbStruct(probStruct)
```

**Exercise 4.13.** Consider again the two-tanks setup from the previous two exercises. Finally we will synthesize an MPC controller. To do that, run

```
>> ctrl = mpt_control(sysStruct, probStruct, 'online')
```

The `'online'` flag indicates that we will use an on-line MPC controller. After the controller object is constructed, we can obtain the optimal control input associated to a given initial state by running

```
>> uopt = ctrl(x)
```

Now answer following questions:

1. What is the optimal control action for  $x(0) = \begin{bmatrix} -5 \\ -10 \end{bmatrix}$ ?
2. Which input corresponds to  $x(0) = \begin{bmatrix} 0 \\ 12.5 \end{bmatrix}$ ? Can you give the answer even before running the code in MATLAB? (hint: take a look at state constraints in `sysStruct.xmax`)
3. Plot the closed-loop evolution of system states and control inputs for 10 steps, starting from the initial condition  $x(0) = \begin{bmatrix} -5 \\ -10 \end{bmatrix}$ .

**Exercise 4.14.** Assume again the setup from the previous exercise, but change the `probStruct.R` penalty to 10. Run again the closed-loop simulation and explain what has changed.

We also have a good news for you. MPT provides a simple command to visualize a closed-loop simulation:

```
simplot(ctrl, x0, number_of_simulation_steps)
```

If you only want to compute the numerical data, use the `sim` command:

```
[X, U, Y] = sim(ctrl, x0, number_of_simulation_steps)
```

**Exercise 4.15.** Now we will practice with explicit MPC controllers. In the explicit approach the MPC problem is “pre-solved” and the optimal control action is computed *for all* possible initial conditions. The solution then takes a form of a look-up table, which consists of polyhedral regions with associated affine feedback laws. Evaluation of such controllers then reduces to a mere table lookup, which can be performed much faster compared to on-line controllers.

We consider control of the double integrator, described by the following state-space realization:

$$\begin{aligned} x_{k+1} &= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} u_k \\ y_k &= \begin{bmatrix} 1 & 0 \end{bmatrix} x_k \end{aligned}$$

which is subject to constraints  $-1 \leq u_k \leq 1$  and  $\begin{bmatrix} -5 \\ -5 \end{bmatrix} \leq x_k \leq \begin{bmatrix} 5 \\ 5 \end{bmatrix}$ .

First step for you is to create the corresponding `sysStruct` structure for such a system (make sure to run `clear` to wipe the workspace). When you have that, create the `probStruct` structure, assuming the following assignment:

- prediction horizon 5
- quadratic type of the performance objective
- $Q = I_2$  and  $R = 1$

Finally, create two controller objects, one called `controller_online` and representing an on-line MPC controller, and the other one `controller_explicit` where the solution will be pre-calculated.

Hint: use either `'online'` or `'explicit'` string flags as the third input to `mpt_control()`.

Once you have both controllers, verify that they are indeed equivalent. Which control action does `controller_online` give for  $x_0 = [-3; 0]$ ? And what is the value of `controller_explicit` for the same state? Of course, the two values should be identical, since the two controllers represent the same optimization problem.

The only difference is in the evaluation speed. To see it in a more prominent fashion, fire up closed-loop simulations for both controllers using the `sim()` command and with  $x_0 = [-3; 0]$  as the initial state. In both cases measure the execution time using `tic` and `toc` functions. What do you see? (keep in mind that this will measure the execution time of the whole closed-loop simulation. More interesting is the average execution time of one simulation step.)

**Exercise 4.16.** Probably the best way to understand an explicit MPC controller is to visualize it. We just quickly remind that such a controller consists of several polytopic regions with associated affine feedback laws of the form  $u = K_i x + L_i$ .

We start by inspecting the controller's regions:

```
close all
plot(controller_explicit)
```

This figure tells us two important properties of the underlying MPC formulation. First, we see how many regions there are in total. Quite intuitively, the higher number of regions, the more complex the MPC problem is and the more computationally demanding is its implementation in real time. But even more important is the portion of the colored area compared to the white space surrounding it. Any point from within of the colored patches is a *feasible* initial point, i.e. one for which there exists a solution satisfying constraints *for the whole* prediction horizon  $N$ . On the other hand, any point from the white areas is *infeasible*. Again, this information is quite important for real-time implementation, as you can easily reject “bad” initial conditions.

Let's verify these statements by running `u = controller_explicit(x0)` once for  $x_0 = [-4; -1]$  and the other time for  $x_0 = [-4; -2]$ . Do the results support our claim about feasibility? What would happen if we use `u = controller_online(x0)` instead? Can you give an answer even prior to running the command?

Now we move to plotting the associated feedback laws by

```
close all
plotu(controller_explicit)
```

Again, such a graphical information is valuable to control engineers, because it gives us an insights into the “internals” of the optimal solution to a given MPC optimization problem. Specifically, we can see for which range of initial conditions the controller responds by a saturated control action. To see this in an even more clear way, run the `view(2)` command to watch the result “from the top”. You will see dark blue regions which represent all initial conditions for which the optimal control action is saturated at  $u = u_{\min} = -1$ , dark red ones where  $u = u_{\max} = 1$ , and the rainbow stripe in between with  $-1 < u < 1$ .

**Exercise 4.17.** Visualization is not the only way to analyze explicit MPC controllers. MPT provides a wide range of analysis functions. For instance, one can try to compute a Piecewise Quadratic Lyapunov function as a certificate that the controller provides closed-loop stability guarantees:

```
>> L = mpt_lyapunov(controller_explicit, 'pwq');
```

When you run this command, you should see the following output:

```
mpt_getPWQLyapFct: Partition is not invariant and
                    therefore it cannot be asymptotically stable !!
```

The reason for this is that the controller is not invariant. We have already seen what invariance means in Example 6 of the *MPT Polytope Library* part of this workshop. In simple terms, the problem is that there are some initial states for which there exists a feasible control move at the beginning, e.g.

```
>> x = [-5; 4]
>> u = controller_explicit(x)
```

```
u =
```

```
    -1
```

but then the successor state  $x^+ = Ax + Bu$  becomes infeasible:

```
>> xp = model.A*x + model.B*u
```

```
xp =
```

```
   -2.0000
    3.5000
```

```
>> u = controller_explicit(xp)
MPT_GETINPUT: NO REGION FOUND FOR STATE x = [-2;3.5]
```

```
u =
```

```
[]
```

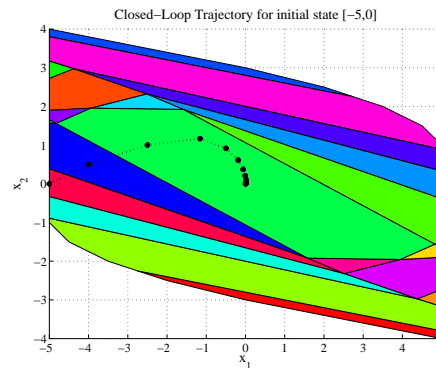
To see this graphically, run

```
>> x = [-5; 4];
>> simplot(controller_explicit, struct('x0', x))
```

You will see that the closed-loop trajectory leaves the colored area, which means that constraint satisfaction is not guaranteed for all time for this initial point.

On the other hand, the state  $x = [-5; 0]$  belongs to the invariant subset, as shown by the existence of the following closed-loop simulation, also shown on the figure below:

```
>> x = [-5; 0];
>> simplot(controller_explicit, struct('x0', x))
```



**Fig. 4.27** Closed-loop simulation illustrating that the state  $x = [-5; 0]$  belongs to the invariant subset.

To isolate the invariant subset of the given explicit controller, run

```
>> inv_controller = mpt_invariantSet(controller_explicit)
```

Ok, but what is the difference between the two controllers? Specifically, which part of `controller_explicit` was NOT invariant (i.e. which subset of regions of `controller_explicit` is not contained in the regions of `inv_controller`)? To see that, we compute the set difference between the regions of the two controllers:

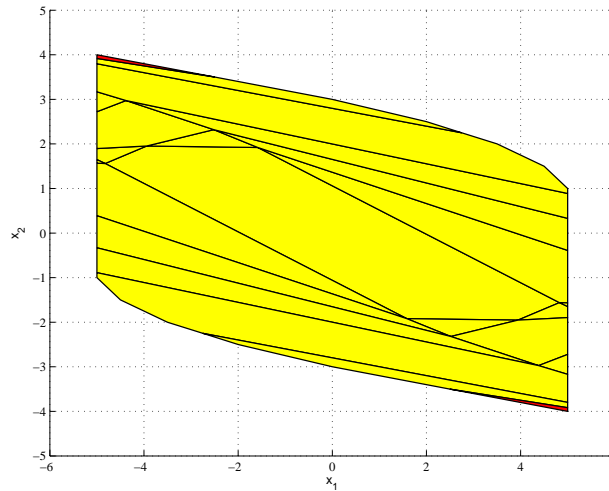
```
>> NI = controller_explicit.Pn \ inv_controller.Pn;
```

and we plot it:

```
% plot regions of the invariant subset in yellow
```

```
% and the states which are not invariant in red
>> plot(inv_controller.Pn, 'y', NI, 'r')
```

You should see the figure similar to the one below.



**Fig. 4.28** Set of invariant states (yellow) and those which are not invariant (the two small red patches in the upper-left and lower-right corners).

With the invariant controller available in `inv_controller`, we can return back to stability analysis:

```
>> L = mpt_lyapunov(inv_controller, 'pwq');
```

This time the Lyapunov function is found:

```
mpt_getPWQLyapFct: SUCCESS:
Found Piecewise Quadratic Lyapunov function.
```

which shows that the explicit controller `inv_controller` will definitely render the closed-loop system stable. You can check this fact by running a bunch of closed-loop simulations using the point-and-click interface:

```
>> simplot(inv_controller)
```

Since the controller is invariant and guarantees stability, do you expect to find a starting point for which the simulation would fail?



## 4.7 Solutions

**Solution 4.11 (for Exercise 4.11).**

```

% it's always wise to wipe the sysStruct variable
clear sysStruct

% plant dynamics
sysStruct.A = [-0.0315, 0; 0.0315, -0.0315];
sysStruct.B = [0.0769; 0];
sysStruct.C = [0 1];
sysStruct.D = 0; % must be specified even if it is zero

% constraints
sysStruct.xmin = [-21; -21];
sysStruct.xmax = [3.5; 3.5];
sysStruct.umin = -17;
sysStruct.umax = 3;
sysStruct.ymin = -21;
sysStruct.ymax = 3.5;

% symbolic labels
sysStruct.StateName = {'level_1', 'level_2'};
sysStruct.InputName = {'inflow'};
sysStruct.OutputName = {'level_2'};

% verify the setup
mpt_verifySysStruct(sysStruct);

% simulate the evolution for 5 steps
x = [-5; -10]; % initial state
u = -2*ones(10, 1); % inputs to use in the simulation
X = x'; % store of the simulated states
for k = 1:5
    x = mpt_simSys(sysStruct, x, u(k));
    X = [X; x];
    x = x';
end

% plot the state trajectories
t = 0:size(X, 1)-1;
plot(t, X)

```

**Solution 4.12 (for Exercise 4.12).**

```

probStruct.N = 6;

```

```

probStruct.Q = eye(2);
probStruct.R = 1;
probStruct.norm = 2;
mpt_verifyProbStruct(probStruct);

```

**Solution 4.13 (for Exercise 4.13).**

1. What is the optimal control action for  $x(0) = \begin{bmatrix} -5 \\ -10 \end{bmatrix}$ ?

```

% we assume sysStruct and probStruct have been defined
ctrl = mpt_control(sysStruct, probStruct, 'online');

```

```

% optimal "u" associated to x0 = [-5; -10]:
u = ctrl([-5; -10])

```

```

u =
    -0.0121

```

2. Which input corresponds to  $x(0) = \begin{bmatrix} 0 \\ 12.5 \end{bmatrix}$ ? Can you give the answer even before running the code in MATLAB? (hint: take a look at state constraints in `sysStruct.xmax`)

```

u = ctrl([0; 12.5])

```

```

u =
    NaN

```

Here, the NaN output indicates that the MPC problem was infeasible for a given initial condition. In fact, the second element of  $x(0)$  violates the second state constraint:

```

[0; 12.5] <= sysStruct.umax

```

3. Plot the closed-loop evolution of system states and control inputs for 10 steps, starting from the initial condition  $x(0) = \begin{bmatrix} -5 \\ -10 \end{bmatrix}$ .

```

% closed-loop simulation:
x = [-5; -10]; % initial state
X = x';        % store for closed-loop states
U = [];        % store for closed-loop inputs
for k = 1:5
    u = ctrl(x); % obtain optimal control action
    x = mpt_simSys(sysStruct, x, u); % simulate the system
    X = [X; x];
    U = [U; u];
    x = x';
end

```

```

% plot the results

```

```

tx = 0:size(X, 1)-1;
figure; plot(tx, X);
tu = 0:size(U, 1)-1;
figure; stairs(tu, U);

```

**Solution 4.14 (for Exercise 4.14).** By enlarging the input penalty, the regulation process is slowed down, i.e. it takes longer for the states to reach the zero levels. Conversely, by enlarging the `probStruct.Q` penalty, the controller will react more aggressively.

**Solution 4.15 (for Exercise 4.15).**

```

clear

% prediction sysStruct for the double integrator
sysStruct.A = [1, 1; 0, 1];
sysStruct.B = [1; 0.5];
sysStruct.C = [1 0];
sysStruct.D = 0;

% constraints
sysStruct.umin = -1;
sysStruct.umax = 1;
sysStruct.xmin = [-5; -5];
sysStruct.xmax = [5; 5];

% objective function
probStruct.N = 5;
probStruct.norm = 2;
probStruct.Q = eye(2);
probStruct.R = 1;

% on-line controller
controller_online = mpt_control(sysStruct, probStruct, '
    online');

% explicit controller
controller_explicit = mpt_control(sysStruct, probStruct, '
    explicit');

% initial state for the closed-loop simulations
x0 = [-3; 0];

% timing of the on-line controller
tic;
X = sim(controller_online, x0);

```

```

t = toc; t/size(X, 1)

% timing of the explicit controller
tic;
X = sim(controller_explicit, x0);
t = toc; t/size(X, 1)

```

**Solution 4.16 (for Exercise 4.16).**

```

% run ex_2 to load all necessary data

close all

% plot the controller regions
plot(controller_explicit)

% plot the two initial conditions of interest
x1 = [-4; -1];
x2 = [-4; -2];
hold on
plot(x1(1), x1(2), 'kx', x2(1), x2(2), 'ko', 'markersize',
     12);

% optimal control action associated to x1
%
% since x1 is contained in one of the regions,
% we expect a feasible answer
u = controller_explicit(x1)
% check that x1 is indeed in one of the regions
isinside(controller_explicit.Pn, x1)

% optimal control action associated to x2
%
% since x2 is outside of the colored area,
% there should be no control action associated
% to this state, in which case u = []
u = controller_explicit(x2)
% check that x2 is indeed outside of the regions
isinside(controller_explicit.Pn, x2)

% now plot the feedback laws
close all
plotu(controller_explicit)

% rotate the graph manually to inspect it

```

```
% finally, look at it from the top  
view(2)
```

**Acknowledgements** The authors are pleased to acknowledge the financial support of the Scientific Grant Agency of the Slovak Republic under the grants 1/0071/09 and 1/0537/10 and of the Slovak Research and Development Agency under the contracts No. VV-0029-07 and No. LPP-0092-07. It is also supported by a grant No. NIL-I-007-d from Iceland, Liechtenstein and Norway through the EEA Financial Mechanism and the Norwegian Financial Mechanism.

## References

- Borrelli F (2003) Constrained Optimal Control Of Linear And Hybrid Systems, Lecture Notes in Control and Information Sciences, vol 290. Springer
- Geyer T (2005) Low Complexity Model Predictive Control in Power Electronics and Power Systems. Dr. sc. thesis, ETH Zurich, Zurich, Switzerland, available from <http://control.ee.ethz.ch/index.cgi?page=publications;action=details;id=2124>
- Kvasnica M, Fikar M, Čírka L, Hecceg M (2011a) Complexity reduction in explicit model predictive control. In: Huba M, Skogestad S, Fikar M, Hovd M, Johansen TA, Rohal-Ilkiv B (eds) Selected Topics on Constrained and Nonlinear Control. Textbook, STU Bratislava – NTNU Trondheim, pp 241–288
- Kvasnica M, Rauová I, Fikar M (2011b) Separation functions used in simplification of explicit mpc feedback laws. In: Huba M, Skogestad S, Fikar M, Hovd M, Johansen TA, Rohal-Ilkiv B (eds) Preprints of the NIL workshop: Selected Topics on Constrained and Nonlinear Control, STU Bratislava – NTNU Trondheim, pp 48–53

*Comments – Remarks*

*Comments – Remarks*

*Comments – Remarks*



*Comments – Remarks*

# Chapter 5

## Implementation of MPC Techniques to Real Mechatronic Systems

Gergely Takács and Tomáš Polóni and Boris Rohal'-Ilkiv and Peter Šimončič and Marek Honek and Matúš Kopačka and Jozef Csambál and Slavomír Wojnar

**Abstract** This chapter is focused on the implementation details of model predictive control for mechatronic systems with fast dynamics. From the very

---

Gergely Takács

Institute of Measurement, Automation and Informatics, Faculty of Mechanical Engineering, Slovak University of Technology in Bratislava, e-mail: [gergely.takacs@stuba.sk](mailto:gergely.takacs@stuba.sk)

Tomáš Polóni

Institute of Measurement, Automation and Informatics, Faculty of Mechanical Engineering, Slovak University of Technology in Bratislava, e-mail: [tomas.poloni@stuba.sk](mailto:tomas.poloni@stuba.sk)

Boris Rohal'-Ilkiv

Institute of Measurement, Automation and Informatics, Faculty of Mechanical Engineering, Slovak University of Technology in Bratislava, e-mail: [boris.rohal-ilkiv@stuba.sk](mailto:boris.rohal-ilkiv@stuba.sk)

Peter Šimončič

Institute of Measurement, Automation and Informatics, Faculty of Mechanical Engineering, Slovak University of Technology in Bratislava, e-mail: [peter.simoncic@stuba.sk](mailto:peter.simoncic@stuba.sk)

Marek Honek

Institute of Measurement, Automation and Informatics, Faculty of Mechanical Engineering, Slovak University of Technology in Bratislava, e-mail: [marek.honek@stuba.sk](mailto:marek.honek@stuba.sk)

Matúš Kopačka

Institute of Measurement, Automation and Informatics, Faculty of Mechanical Engineering, Slovak University of Technology in Bratislava, e-mail: [matus.kopacka@stuba.sk](mailto:matus.kopacka@stuba.sk)

Jozef Csambál

Institute of Measurement, Automation and Informatics, Faculty of Mechanical Engineering, Slovak University of Technology in Bratislava, e-mail: [jozef.csambal@stuba.sk](mailto:jozef.csambal@stuba.sk)

Slavomír Wojnar

Institute of Measurement, Automation and Informatics, Faculty of Mechanical Engineering, Slovak University of Technology in Bratislava, e-mail: [slawomir.wojnar@stuba.sk](mailto:slawomir.wojnar@stuba.sk)

wide application possibilities of MPC and the numerous examples of mechatronic systems, two real-life laboratory examples have been selected. First an active vibration attenuation system is discussed, with numerous hardware implementation details. This is then followed by a detailed discussion of the implementation of three very different stabilizing MPC approaches: traditional quadratic programming based MPC, pre-computed explicit MPC and a sub-optimal approach called Newton-Raphson's MPC. Details are given on software and code implementation using coding examples in the Matlab and C programming environment. The next section of this chapter deals with a very different practical example: a spark ignition engine example. This section will introduce the details of a full laboratory setup, hardware implementation details and practical issues regarding the use of MPC for the real-time control of air-fuel ratio in automotive engines.

## 5.1 Introduction

The merits of well designed control algorithms have been long recognized by the industry, where a performance improvement of a fraction point leads to significantly increased revenue or decreased production cost.

Performance is not the only criterion to judge the qualities of a control strategy. Every real actuator and process has inherent physical limits, which are to be constrained in order to preserve safety, further influence production output or to safeguard equipment. As claimed by [Rossiter \(2003\)](#), the only control method capable of handling process constraints on an algorithmic level is model predictive control (MPC). MPC has been used for decades in applications, where sampling rates are in the order of several minutes or even hours: for example in the petrochemical industry. Now the situation turns to applications with very short sampling periods in orders of milliseconds mainly in mechatronic systems.

This chapter contributes to experimental implementation and real-time verification of chosen efficient MPC techniques applied for control of fast mechatronic systems, namely laboratory vibration system (subsection 5.2) and internal combustion engine (subsection 5.3).

The section 5.2 discusses the implementation properties of three different MPC approaches on an experimental laboratory model. This device is basically a simple lightly damped mechanical structure. The algorithms considered here include the traditional infinite horizon dual-mode quadratic programming based MPC (QMPC), pre-computed optimal multi-parametric MPC (MPMPC) and the efficient albeit sub-optimal Newton-Raphson's MPC (NRMPC).

The control of automotive internal combustion engines is one of most complex control problems due to nonlinearities and variable time delays. This is reflected in the time variability of the model parameters in different regions

of the engine operating space. The section 5.3 analyzes the practical aspects of real-time implementation of a multi-model MPC technique to cope with the problem of nonlinearities and variable time delays in controlling of the SI engine air-fuel ratio.

In both sections the main practical software and hardware aspects of the suggested real-time solutions are discussed in details.

## 5.2 Implementation of Various MPC Methods for the Vibration Control of Lightly Damped Cantilevers

### 5.2.1 Introduction

The algorithmic support of active vibration suppression applications is usually limited to simple positive position feedback (PPF) or for example strain-rate feedback (SRF) control (Song et al, 2002). Other essential strategies include PID or linear-quadratic (LQ) control (Preumont, 2002). Because of the simplicity of these approaches, no issues regarding computational implementability arise. Even though these control schemes are often sufficient in certain applications, the advantages of MPC cannot be overlooked in vibration damping.

Unfortunately the fast dynamics of vibration attenuation systems require very short sampling times, which limit the use of computationally intensive on-line calculations usually associated with MPC. Traditional quadratic programming based MPC (QPMP) has been successfully implemented by Wills et al (2008), where the problems raised by the fast sampling were tackled by implementing a machine code optimized quadratic programming solver on a specialized hardware. While this algorithm handles process constraints, it fails to address the important consequences of stability and constraint feasibility.

The stability and constraint feasibility of model predictive control algorithms may be guaranteed through the deployment of terminal constraints (Maciejowski, 2002). This unfortunately reduces the range in which the control algorithm may operate, by assigning a region of attraction of all feasible initial conditions in the state-space. Amongst others, the size of this region is dependent on the prediction horizon.

Clamped cantilever beams actuated by piezoelectric strips are a very special case in the eye of the control engineer. Model predictive control with guaranteed constraint stability and feasibility applied on such and similar systems raises not only questions associated with algorithm speed but also with minimal useful prediction horizon length. Due to the stability requirement, the useful range the stable MPC controller can operate is severely lim-

ited, calling for extremely long prediction horizons (Takács and Rohal'-Ilkiv, 2009).

This is partly caused by the large relative deflections compared to the maximal possible control effort by the piezoelectric actuators, and short sampling times. Optimization of prediction dynamics introduced by Cannon and Kouvaritakis (2005) offers a remedy to this problem, as it maximizes the region of attraction which defines the useful controller range, however this idea is executed for a computationally efficient albeit sub-optimal MPC approach called Newton-Raphson's MPC.

This subsection describes the practical implementation of different MPC methods onto systems of active vibration control with under-damped dynamics. Three MPC methods are introduced here:

- traditional dual-mode quadratic-programming based MPC (QPMP) with stability and feasibility guarantees
- a computationally efficient sub-optimal control strategy introduced by Kouvaritakis et al (2000), Kouvaritakis et al (2002) and Cannon and Kouvaritakis (2005), called Newton-Raphson MPC (NRMPC)
- and finally an optimal multi-parametric programming based MPC approach (MPMPC)

This section places focus on practical implementation details, while suggesting the simplest possible solutions. Therefore no customized on-line quadratic programming solvers or algorithms of multi-parametric programming are discussed here. All methods will be implemented in Matlab/Simulink, if possible using off the shelf solutions for off and on-line solvers. Typically the problem setup will be carried out in Matlab, while the on-line problem is running on an xPC Target platform system. We will begin our discussion with introducing the experimental system used as an example.

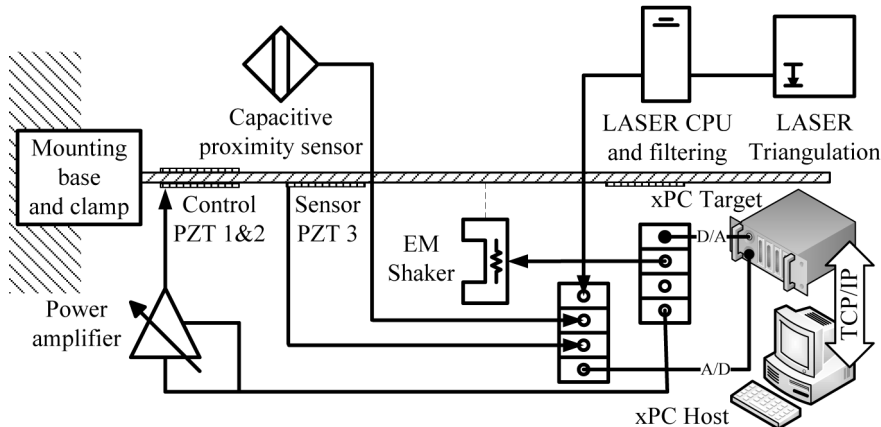
### ***5.2.2 Brief Description of the Experimental Hardware***

A clamped cantilever beam may represent many real life under-damped flexible structures, such as helicopter rotor beams, antenna masts, manipulating arms and solar panels on space structures. The experimental setup considered as an example to demonstrate different MPC implementations may represent the vibration attenuation of any of the aforementioned physical systems.

The aim of the MPC based vibration control system is to minimize beam tip vibrations, while respecting constraints set by the maximal allowable voltage levels on the piezoelectric patches. Since piezoelectric materials are prone to depolarization, maximal voltage input levels are set. Additional requirements placed on the control system are the need for guaranteed stability and constraint feasibility.

### 5.2.2.1 Hardware Description

The beam is made of aluminium marked by the designation EN AW 1050A. Its dimensions are  $550 \times 40 \times 3$  mm. The four piezoelectric patches bonded to the beam surface are identical, of the make MIDÉ QP16. Wafers marked with PZT1/2 in Fig. 5.1 are connected counter phase and are used in actuator mode for control. The rest of the patches are short circuited and are not utilized in the upcoming implementation examples.

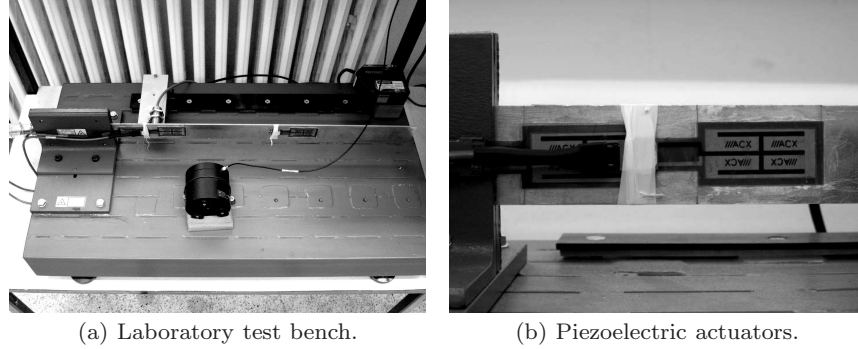


**Fig. 5.1** Simplified schematics of the control and measuring chain applied to the vibrating clamped cantilever beam.

Piezoelectric actuators are connected to 20V/V gain MIDÉ EL-1225 power amplifiers, which in turn receive low level analogue voltage input from a cable terminal fed to a high speed National Instruments PCI-6030E measuring card. This measuring card is a part of a computer running Matlab xPC Target real time control environment. Development of the control algorithms and measuring tasks is carried out using a host computer running Matlab Simulink, and is connected to the target computer via Ethernet.

Beam tip deflection is measured using a KEYENCE LK - G 82 industrial LASER triangulation system, indicated as “Laser triangulation” on the figure. Digital low band pass filtering and signal processing is realized in a proprietary unit, which outputs an analogue voltage signal to the cable terminal and finally the measuring card. The LASER system settings are controlled via the host computer, using a proprietary settings software via USB protocol. The control and measuring chain is demonstrated by the simplified schematics in Fig. 5.1.

Note that the capacitive sensor is not utilized as a feedback signal in these examples, a direct readout using the laser triangulation system is considered.



**Fig. 5.2** A photo of the laboratory test bench with the piezoelectric actuators bonded on the beam.

A photo of the laboratory test bench with the piezoelectric actuators bonded on the beam is visible In Fig. 5.2.

### 5.2.2.2 System Identification

To predict future system behaviour, a second order linear-time invariant state-space mathematical model of the vibrating system is considered. To obtain this model, [Lara et al \(2003\)](#) used a direct phenomenological model of the system, or for example it is possible to utilize the results of finite element modeling (FEM) harmonic analysis to acquire the state-space representation as well ([Dong et al, 2006](#)).

An experimental identification procedure will be utilized to obtain a discrete linear time invariant state-space system in the form:

$$x_{k+1} = Ax_k + Bu_k \quad y_k = Cx_k + Du_k \quad (5.1)$$

where  $y_k$  represents the beam tip deflection in millimeters and  $u_k$  is the high level voltage signal directly fed to the piezoelectric actuators. As in most real systems, direct control feed-through is not considered, therefore matrix  $D$  is set to equal zero.

The experimental procedure involved measurements in the frequency domain. The system has been excited by an amplified chirp signal entering the actuators PZT1 and PZT2. The amplitude of this signal has been set to reach maximal allowable levels of  $\pm 120$  V, in order to minimize vibration caused by the surroundings. Since the first eigenfrequency of the beam is 8.127 Hz, the chirp signal frequency started from 0.1 Hz up to 20 Hz. The measured signal has been sampled 5000 Hz, and subsequently re-sampled for different models. The predictions are generated by the following state-space system sampled by 100 Hz:

$$\begin{aligned}
 A &= \begin{bmatrix} 0.9981 & -1.2233 \\ 0.0021 & 0.9985 \end{bmatrix} & B &= \begin{bmatrix} 6.697E^{-6} \\ -8.001E^{-6} \end{bmatrix} \\
 C &= [-0.5774 \quad -0.7069] & & (5.2)
 \end{aligned}$$

### 5.2.3 Quadratic Programming based MPC

#### 5.2.3.1 Introduction

This subsection will elaborate on the practical implementation of a dual-mode quadratic programming based MPC controller with stability and feasibility guarantees. The on-line quadratic programming solver considered in this application is called qpOASES and it has been introduced by [Ferreau et al \(2008\)](#) and subsequently [Ferreau \(2006\)](#). This solution has been chosen for its easy implementation properties and the fact that unlike generic QP solvers, it has been fine tuned for MPC applications. The problem setup, creating prediction and cost prediction matrices is carried out in the Matlab scripting environment. The final problem parameters are passed onto the Simulink interface of the qpOASES solver.

Stability in the traditional QPMPC formulation is guaranteed through suitably formulated state feedback and terminal cost function matrices. The deployment of dual mode predictions is the part of the formulation as well: the first mode considers  $n_c$  free control moves, while the second mode assumes the LQ control law ([Chen and Allgöwer, 1998](#); [Mayne et al, 2000a](#)). Feasibility of process constraints is ensured beyond the prediction horizon by the inclusion of a constraint checking horizon.

First let us begin with the setup of the problem. For this, the code shall be implemented in the Matlab m-script language:

#### 5.2.3.2 Setup

First it is required to specify certain parameters for the QP controller. Some of these parameters are: the sampling period  $T_s$ , if it is an off-line simulation a stop time is required as well, and it is also essential to state the prediction horizon  $n_c$ . In case symmetric bounds on the input are assumed, the constraints are set using *umax*. This is basically the polarization voltage of the piezoelectric transducers.

```

Ts=0.01;
T=0.5
nc=70;
run=T/Ts;

```



```
umax = 120;
```

where *run* is the runtime in case an off-line simulation is needed.

In the next step, it is required to load and specify a prediction model and possibly an initial state for Kalman filtering or simulation purposes. In this example the model is loaded from a saved system identification file:

```
load n4s2A;
A=n4s2.A;
B=n4s2.B;
C=n4s2.C;
nx=length(A);
X1(:,1)=zeros(1,nx);
```

Penalization for the inputs and states needs to be stated as well. The input penalization can be determined by direct experimentation with the algorithm, or simply evaluating different linear-quadratic controllers in simulation and determining a good balance between controller performance and aggressiveness. In this case the input penalty has been found by using an LQ controller with the settings  $R = 1e - 4$  and  $Q = C' * C$ , while balancing the input somewhat above the constraints.

```
R=1e-4;
Q=C'*C;
```

### 5.2.3.3 Prediction Matrices

It is possible to pass the information to a stand-alone function. This custom function uses the system model, penalty matrices, the constraints, a horizon and possibly a system order information. Its output are the properly formulated prediction matrices and possibly the re-formulated constraints:

```
[H,F,G,Ac,b0,Bx,Ki,Nc] = predmodelqp(A,B,C,R,Q,umax,nc,nx);
```

Now let us begin with examining what such a function may do, in order to generate the prediction matrices and cost function prediction matrices for the on-line run.

As in most vibration damping applications, this implementation assumes a symmetric constraint on the input:

```
u1=-uh;
```

This is followed by calculating the unconstrained linear-quadratic optimal gain, along with the terminal weighting matrix:

```
[K,S,e]=dlqr(A,B,Q,R);
K=-K;
Qe=dlyap((A+B*K)',(Q+K'*R*K));
```

The forced and free state prediction matrices are calculated through a set of nested loops according to the following script:

```
M2=zeros(nc*nx);
for n=1:nc;
    M1(n*nx-nx+1:n*nx,:)=[(A^n)];
    for na=0:nx:(nc*nx); %na=0:nx:(nc*nx)
        m2(nx*(n-1)+(na+1):nx*(n-1)+(na+nx),n)=[(A^(na/nx))*B];
    end;
end;
M2=m2(1:nx*nc,:);
```

where several other possible solutions may exist. These solutions can be equivalently good, and while their runtime may differ this should not be an issue in an off-line problem setup process.

The last  $n$  rows of the matrices  $M1$  and  $M2$  are also selected:

```
M11=M1(nx*nc-(nx-1):nx*nc,:);
M21=M2(nx*nc-(nx-1):nx*nc,:);
```

The next step is to create the cost prediction matrices. One has to begin with initialization:

```
H1=0;
F1=0;
G1=A^0*Q;
```

This is then followed by creating the cost prediction matrices  $H$ ,  $F$  and  $G$  - first by running the following loop to get partial results:

```
for i=0:nc-2
    H1t=M2(1+i*nx:i*nx+nx,:)'*Q*M2(1+i*nx:i*nx+nx,:);
    H1=H1+H1t;
    F1t=M2(1+i*nx:i*nx+nx,:)'*Q*M1(1+i*nx:i*nx+nx,:);
    F1=F1+F1t;
    G1t=M1(1+i*nx:i*nx+nx,:)'*Q*M1(1+i*nx:i*nx+nx,:);
    G1=G1+G1t;
end
```

And finally assembling cost prediction matrices  $H$ ,  $F$  and  $G$ :

```
H=H1+M2(1+(nc-1)*nx:(nc-1)*nx+nx,:)'*Qe*M2(1+(nc-1)*nx:(nc-1)*
    nx+nx,:)+R*eye(nc);
F=F1+M2(1+(nc-1)*nx:(nc-1)*nx+nx,:)'*Qe*M1(1+(nc-1)*nx:(nc-1)*
    nx+nx,:);
G=G1+M1(1+(nc-1)*nx:(nc-1)*nx+nx,:)'*Qe*M1(1+(nc-1)*nx:(nc-1)*
    nx+nx,:);
```

To ensure feasibility and stability beyond the prediction horizon, the constraint checking horizon is calculated as well. This process is started up by an initialization procedure:

```

Ki=K;
Ki(2,:)=K*(A+B*K);
i=1;
Nc=0;
u=uh+1;

```

The length of the constraint checking horizon is computed in the following loop:

```

while (u > uh);
    Ki(i+2,:)=K*(A+B*K)^(i+1);
    f=Ki(i+2,:);
    Am=[Ki(1:(i+1),:);-Ki(1:(i+1),:)];
    b=[uh*ones((i+1),1); -ul*ones((i+1),1)];
    x0=linprog(-f,Am,b);
    u=Ki(i+2,:)*x0;
    Nc=Nc+1;
    i=i+1;
end

```

This can be followed by defining the constraints and re-formulating them to be useful for direct quadratic programming solution. This formulation assumes symmetric input constraints:

```

Ac1=[eye(nc)];
b0=[uh*ones(nc+Nc,1); -ul*ones(nc+Nc,1)];
Bx1=zeros((nc-1),nx);

for i=0:Nc
    Ac1(nc+i,:)=Ki(i+1,:)*M21;
    Bx1(nc+i,:)=[-Ki(i+1,:)*M11];
end

Ac=[Ac1;-Ac1];
Bx=[Bx1; -Bx1];

```

#### 5.2.3.4 Re-formulating for the Simulink Interface

This sub-subsection introduces a way to reformulate the prediction matrices, so they can be used directly with the Simulink interface of qpOASES. First the cost prediction matrix  $H$  is re-formulated, so it is suitable to pass on to the qpOASES problem:

```

Hqp=[];
for i=1:nc
    Hqpt=H(i,:);

```

```
Hqp=[Hqp Hqpt];
end
```

Passing on  $F$  is possible with the original formulation. The cost matrix  $Ac$  needs to be transformed likewise:

```
AcQP=Ac(1:nc+Nc,:);
AcQP=[];
for i=1:(nc+Nc)
    AcQPt=AcQP(i,:);
    AcQP=[AcQP AcQPt];
end
```

where the matrices need to be divided in the usual C programming style, along with constraint matrices  $Bx$  and  $b0$ :

```
BxQP=Bx(1:nc+Nc,:);
b0QP=b0(1:nc+Nc,:);
```

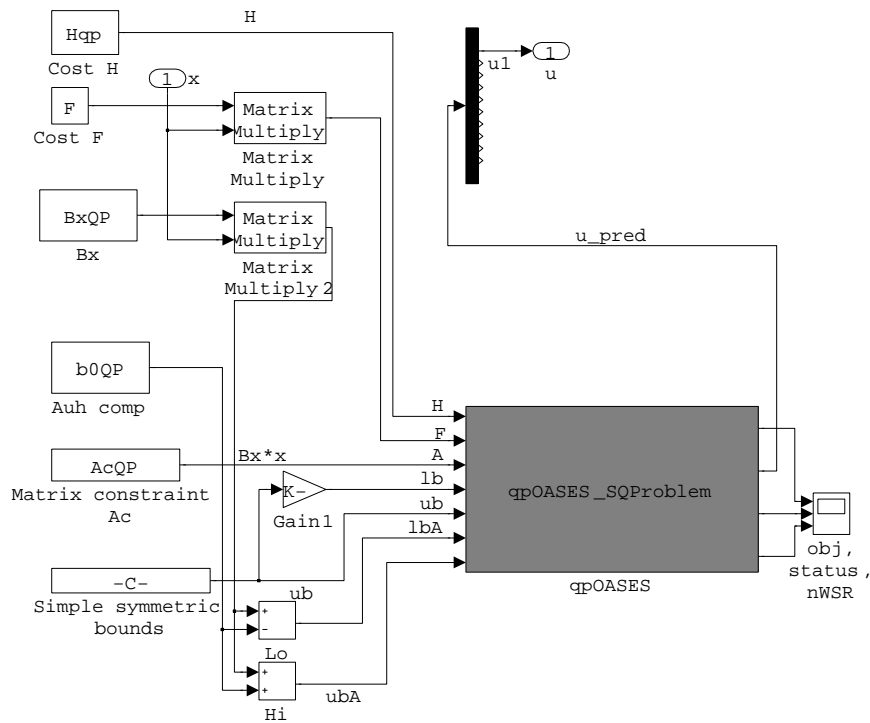


Fig. 5.3 Parsing the MPC problem to the qpOASES interface

### 5.2.3.5 Off-line Simulation

In case an off-line simulation is necessary for results verification or other purposes, we may use the Matlab default quadratic solver, named *quadprog*. To do this, one needs to launch a cycle with one iteration for each sampling instant and supply the optimization problem to the solver in the following fashion:

```
for k=1:run;
    [U1(:,k),f,status1(1,k),output]=quadprog(H,F*(X1(:,k)),Ac,b0+
        Bx*X1(:,k),[],[],[],[],[],options);
    X1(:,k+1)=A*X1(:,k)+B*(U1(1,k))';
    Y1(k)=C*X1(:,k);
end
```

where  $X1$  is the matrix containing the states and  $Y1$  is the vector containing the deflection data. The last two lines assume that there is no need for state observation, this is to make the simulation simpler.

If the cost is needed as well, one needs to include either one of the following lines in the code:

```
J(k,:)= U1(:,k)'*H*U1(:,k)+2*(X1(:,k))'*F'*U1(:,k)+X1(:,k)'*G
        *X1(:,k);
J2(k,:)= X1(:,k)'*Q*X1(:,k)+U1(1,k)'*R*U1(1,k);
```

The optimization procedure can be fine-tuned by:

```
options = optimset('LargeScale','off','Display','off','TolFun',
    ,1e-12);
```

It is also possible to substitute the Matlab built-in *quadprog* function with the qpOASES Matlab interface. After compilation the sequential qpOASES solver can be simply called by using the following code:

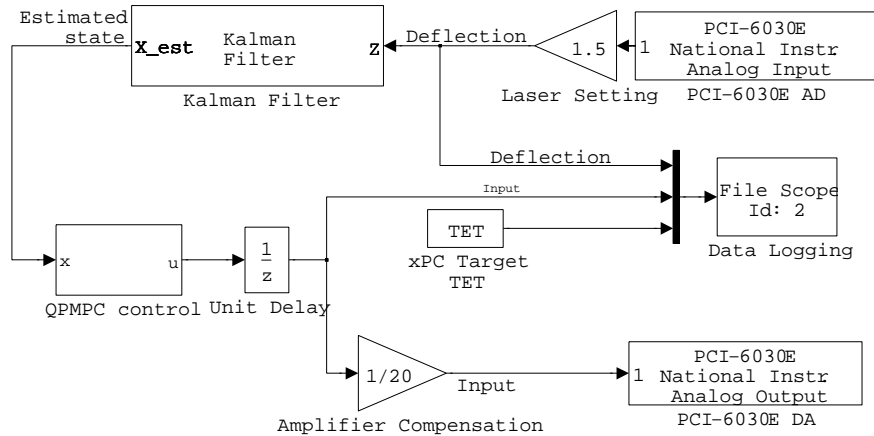
```
for k=1:run;
    [obj0A,U2(:,k),yd,status2(1,k),nWSRout2(1,k)]=
        qpOASES_sequence('i',H,F*X2(:,k),Ac','','','','b0+Bx*X2(:,
        k),10);
    X2(:,k+1)=A*X2(:,k)+B*U2(1,k);
    Y2(k)=C*X2(:,k);
end
```

### 5.2.3.6 Controller Loop in Simulink

The QPMPc controller loop is featured in Fig. 5.3. The controller loop contains means for A/D and D/A conversion, state observation, data logging and others. The block containing the QPMPc controller takes the actual observer state as an input, and outputs controller voltage. It may be required to

output such variables as iteration data or cost function, but that is reserved for diagnostic purposes. The state observation in this example is carried out via a product-default Kalman filter.

The subsystem marked as QPMPC in Fig. 5.3 is featured in its full detail in Fig. 5.4. Here it is visible how the predicted cost matrices calculated in the problem setup stage are parsed onto the qpOASES interface. One shall refer to the qpOASES documentation for details.



**Fig. 5.4** Block scheme of the QPMPC controller in Simulink, for the active vibration cancellation application

## 5.2.4 Newton-Raphson's Suboptimal MPC

### 5.2.4.1 Introduction

This subsection will introduce the practical implementation of the Newton-Raphson MPC algorithm into the vibration attenuation of lightly damped structures. As with the other cases, problem setup will be carried out using the Matlab script language, while the on-line part of the algorithm will be implemented in the C language, combined with the xPC Target software prototyping interface.

The computationally efficient NRMPC formulation with guaranteed stability and feasibility of constraints has been first introduced by [Kouvaritakis et al \(2000\)](#). In this approach the on-line optimization task is performed through the Newton-Raphson root search algorithm. Although this is a sub-optimal approach, its application to vibration suppression systems is attractive.

Optimality levels may be enhanced by extending NRMPC by a simple explicit additional optimization, as introduced by Kouvaritakis et al (2002). The further development of NRMPC introduced by Cannon and Kouvaritakis (2005) proves to be essential for systems with a significant asymmetry in actuator capabilities and deformation range. The optimization of prediction dynamics formulated by Cannon and Kouvaritakis (2005) recovers the maximal possible ellipsoidal region of attraction.

The algorithm described by Kouvaritakis et al (2000), Kouvaritakis et al (2002) and Cannon and Kouvaritakis (2005) serves as a basis for the practical implementation applied to the active vibrating system. For more details on the theory behind the NRMPC approach, the reader shall refer to the aforementioned publications.

Let us begin with initializing the off-line part of the NRMPC code:

#### 5.2.4.2 Initialization of the Off-line Code

The controller structure is obtained by evaluating linear matrix inequalities (LMI) defined by the invariance and feasibility conditions. LMI are parsed using YALMIP introduced by Lofberg (2004) to the SeDuMi optimization software suite as formulated by Sturm (1999).

The first portion of the off-line code initializes the algorithm. Amongst others, simulation stop time is defined along with the desired deflection, which in the case of the vibration attenuation example is always zero. A linear time-invariant state space model is loaded from a file. Sampling time is also defined as  $T_s$ :

```
Tstop=60;
yd=0;
load m2ss.mat;
A=m2ss.A; B=m2ss.B; C=m2ss.C; D=m2ss.D;
Ts=0.01;
```

Other types of settings and tuning parameters are also declared at the beginning of the script file. Symmetric input constraints are stated as  $uh$ . State penalties are set as  $Q = C^T C$  which includes the output deflection in the computed cost. Input weighting is declared as the variable  $R$ :

```
uh=120;
Q=C'*C;
R=1e-4;
```

Prediction cost performance bound  $\gamma$  is stated, which is necessary to be limited in order to preserve numerical stability of the process. A tolerance limit is also set, which is used to modify the behavior of YALMIP regarding the handling of strict inequalities.

```
tolerance=1e-8;
```

```
gamma=1e6;
```

Dimensionality of the problem is determined, to allow the use of different state-space models for generating predictions:

```
dim=size(An); dim=dim(1,1);
[w.dim]=size(B); w.dim(1,3)=w.dim(1,1);
w.dim(1,4)=w.dim(1,2)*w.dim(1,3);
w.dim(1,5)=size(Q,1);
```

Matrix square-roots of the penalization variables are computed. These are required in the construction of the invariance condition. The closed loop linear quadratic gain is calculated as well, and it makes use of the prediction model and penalties introduced earlier:

```
sqrtR = sqrtm(R); sqrtQ = sqrtm(Q);
K=-dlqr(An,B,Q,sqrtR*sqrtR);
Phi0=(An+B*K);
```

#### 5.2.4.3 Off-line Variables and Constraints

Four optimization variables are declared, according to the dimensionality of the problem. Matrix  $N$  is fully parametrized and square, while the rest of the optimization variables are real valued and symmetric:

```
Xq = sdpvar(w.dim(1,1),w.dim(1,1));
Yq = sdpvar(w.dim(1,1),w.dim(1,1));
N = sdpvar(w.dim(1,1),w.dim(1,1),'full');
M = sdpvar(w.dim(1,2),w.dim(1,1));
```

The LMI constrain the semi-definite programming problem. The *set* command instructs the parser YALMIP to construct a constraint in an LMI form:

```
Inv1 = [Yq,Xq;Xq,Xq];
Inv3 = [Phi0*Yq+B*M Phi0*Xq;N+Phi0*Yq+B*M Phi0*Xq];
if (gamma<1/tolerance)
    gInv=gamma*eye(w.dim(1,5)+w.dim(1,2));
    zInv=zeros(w.dim(1,5)+w.dim(1,2),2*w.dim(1,1));
    Inv2=blkdiag(sqrtQ,sqrtR)*[Yq,Xq; K*Yq+M,K*Xq];
    F = set([gInv,zInv,Inv2; zInv',
            Inv1,Inv3; Inv2',Inv3',Inv1] > 0);
else
    F = set([Inv1 Inv3; Inv3' Inv1] > 0);
end
```

The *if* construct checks whether there is an input constraint defined or not. If yes, the feasibility condition is translated to the proper LMI and added to



the set of constraints defining the SDP problem. Input constraints are defined by:

```
if ~isempty(uh)
F = F + set([uh^2 [K*Yq+M K*Xq];
[K*Yq+M K*Xq]' Inv1] > 0);
end
```

#### 5.2.4.4 Solver Setup and Off-line Solution Initiation

Options are passed to the LMI parser and also to the solver, in this case SeDuMi. Strict inequality constraints are relaxed and perturbed by the *shift* setting:

```
ops = sdpsettings('verbose',0);
ops = sdpsettings(ops,'shift',10*tolerance);
ops = sdpsettings(ops,'solver','sedumi','sedumi.eps',0)
```

Solution of the above defined SDP problem is initiated by the *solvesdp* YALMIP command. The LMI defining constraints are passed onto the solver as the variable  $F$ , options are contained in the *ops* parameter.

The aim of this optimization problem is to maximize the volume of the ellipsoids defining the region of attraction and target set. This can be carried out by utilizing the fact, that the volume of an ellipsoid is proportional to its determinant:

$$\max \varepsilon = -(\det P)^{1/m} \quad (5.3)$$

where  $P$  is the optimization parameter in general and  $m$  is the dimension of  $P$ . In this case optimization objectives and parameters are  $Yq$  and  $Xq$ , defining the projection and intersection of the augmented ellipsoid into  $x$  space. It is desirable to maximize the volumes of ellipsoids defined by  $Yq$  and  $Xq$  at the same time, by including them in a block diagonal construct.

Optimization parameters  $Yq, Xq, N$  and  $M$  are converted into the standard double precision matrix format, from the YALMIP optimization variable notation:

```
info = solvesdp(F,-geomean(blkdiag(Yq,Xq)),ops);
Yq = double(Yq); Xq = double(Xq);
N = double(N); M = double(M);
```

#### 5.2.4.5 Factoring, Storing and Preparing Parameters for the On-line NRMPC Run

After the optimization variables are available, the parameters used in the on-line NRMPC run have to be factored out and stored:

```
[V,XiU] = lu(eye(w.dim(1,1)) - Yq/Xq);
XiU = XiU';
Qzi = [inv(Xq),XiU;XiU',-V\ (Yq*XiU)];
Qz = [Yq,V;V',-(Xq*XiU)\V];
```

Code segments  $-V\ (Yq*XiU)$  and  $-(Xq*XiU)\ V$  actually implement mathematical operations  $-V^{-1}YqXiU$  and  $-(XqXiU)^{-1}V$ .

The full, optimized shift matrices  $A_0$  and  $C_0$  are calculated according to:

```
A0 = (Xq*XiU)\(N/V');
C0 = M/V';
Q11=inv(Xq)
Q12=XiU;
Q21=XiU';
Q22=-V\ (Yq*XiU);
Kt=[K C0];
```

where code segment  $(Xq*XiU)\(N/V')$  is equivalent to the operation  $(XqXiU)^{-1}KV^{T-1}$ .

A matrix right division is used in the segment  $M/V'$  to implement the operation  $M/V^{T-1}$ . After this respective partitions of  $Q_z$  are stored in variables for the needs of the on-line NRMPC code. It is true that partitions  $Q_{12}$ ,  $Q_{21}$  are related in symmetry.

#### 5.2.4.6 Cost Transformation

The following code segment is related to cost transformation, and the resulting conversion of augmented states. The cost to be minimized in the on-line NRMPC run has been expressed as  $J_{NRMPC} = f^T f$  which can be only true in the case the augmented states are transformed to make the cost equivalent with the original MPC formulation.

```
Mx = dlyap(Phi0',Q+K'*R*K);
Mc = dlyap(A0',C0'*(R+B'*Mx*B)*C0);
```

In order to minimize an equivalent transformed cost and still having the same simple function, the augmented states  $z$  have to be transformed in the on-line optimization task

```
[V,D]=eig(Mc);
d=sqrt(max(diag(D),tolerance));
invT=V*diag(1./d)/V;
invTT=blkdiag(eye(w.dim(1,1)),invT);
```

Select parameters are passed onto the on-line formulation, while some minor practical operations are performed in the final code segment.

```
Pt=invTT'*Qzi*invTT;
```

```
[R,S]=eig(Pt(dim+1:2*dim,dim+1:2*dim));
Sm=diag(S);
Q21=Pt(dim+1:2*dim,1:dim);
```

#### 5.2.4.7 The Newton-Raphson Root Search Algorithm

The first problem in the on-line formulation is to find  $\lambda$ . For this, one needs to use the Newton-Raphson procedure. The underlying concept is very simple (Anstee, 2006) and in relation to the NRMPC procedure it can be stated as:

$$\frac{d\Phi(\lambda)}{d\lambda} = \frac{0 - \Phi(\lambda)}{\lambda_{n-1} - \lambda_n} \quad (5.4)$$

The procedure itself is also trivial and is represented by the following algorithm:

At each sampling instant initialize with  $\lambda = 0$  and perform an iteration which calculates:

$$\lambda_{n+1} = \lambda_n - \frac{\Phi(\lambda)}{\frac{d\Phi(\lambda)}{d\lambda}} \quad (5.5)$$

until the change in  $\lambda$  is smaller than the pre-determined tolerance, where subscript  $n$  denotes the iterations of the Newton-Raphson procedure.

One may take advantage of expressing the matrix  $\hat{Q}_f$  as the an eigenvalue/eigenvector decomposition::

$$\hat{Q}_f^i = RA^iR^T \quad (5.6)$$

where  $R, A$  is defined by the eigenvalue - eigenvector decomposition of  $\hat{Q}_f$  and  $i$  is the  $i$ -th power or inverse. Using the decomposition (5.6) we may denote  $M$  as:

$$M = R \text{diag}(1./(1 - \lambda S_v)) R^T \quad (5.7)$$

where “diag” denotes a diagonalization operation,  $S_v$  is a vector of eigenvalues gained from  $A$  and  $./$  denotes the piecewise division operation. This substitutes the inversion of the full matrix expression  $(I - \lambda \hat{Q}_f)$  in the on-line algorithm. This expression occurs not only in evaluating the perturbation vector  $f$ , but also in the function  $\Phi(\lambda)$  and its first derivative.

Let  $m_v = (1 - \lambda S_v)$  and  $m^i = \text{diag}(1./m_v^{(-i)})$ , then:

$$\Phi(\lambda) = x_k^T W_1 m^2 W_2 x_k + W_3 \quad (5.8)$$

$$\frac{d\Phi(\lambda)}{d\lambda} = 2x_k^T W_4 m^3 W_2 x_k \quad (5.9)$$

The equation yielding the perturbation vector  $f$  will successively transform to

$$f = \lambda R m^1 W_4 x_k \quad (5.10)$$

Matrices  $W_1, W_2, W_3$  can be calculated offline, therefore saving some time avoiding unnecessary multiplications at every NR iteration and sample time. Matrix  $W_4$  can be calculated before the NR process for the actual sample time initiates:

$$\begin{aligned} W_1 &= \hat{Q}_{xf} R \\ W_2 &= W_1^T \\ W_3 &= W_1 A^{-1} \\ W_4 &= -x_k^T W_3 W_2 + x_k^T \hat{Q}_{xx} x_k - 1 \end{aligned}$$

#### 5.2.4.8 Real-time Code Implementation to C

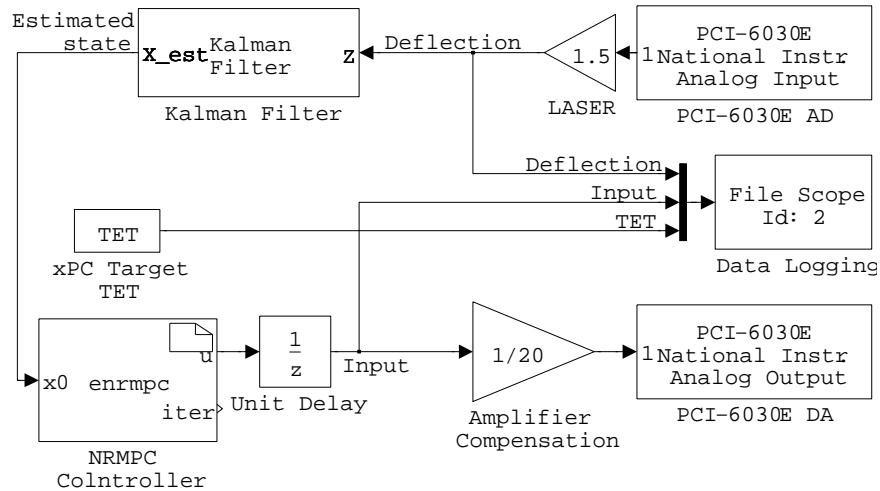
The on-line part of the NRMPC controller is simple. The algorithm does not contain full matrix inversions, only essential piecewise inversions. Matrix and vector operations within the NRMPC on-line algorithm are performed using the Basic Linear Algebra Subprograms (BLAS) package.

The Simulink scheme implementing the NRMPC controller gains its feedback signal directly from a block reading the analogue input of the measurement card. The scheme is visible in Fig. 5.5. The output from this block is scaled according to the current settings of the LASER head, so its output is given directly in millimetres. The File Scope block ensures real-time data logging onto the xPC Target PC hard drive, which can be later re-used for measurement processing.

Feedback measurement passes through a product default Kalman filter, a part of the Signal Processing Blockset toolbox (MathWorks, 2007). The Kalman filter is always enabled, and the number of filters is set to one. Initial condition is assumed to be zero.

The estimated state, conforming to the dimensions of model considered when creating the NRMPC controller, is passed onto a custom S-Function block implementing the on-line NRMPC code in C language. This block takes parameters  $R, T, S_m, Q_{21}, Q_{11}, K$  and  $C_0$  as its set-up and starting parameters. All of these parameters are the result of running the Matlab script responsible for initialization and off-line NRMPC algorithm evaluation. The S-Function block has the core C program and two custom functions (See 5.2.4.9) specified as modules.

The C program is contained within this S-Function, which has been created using the S-Function Builder. In addition to the specified parameters including their data type, input and output ports are declared. Here the input is the estimated state, and output the scalar controller variable. Discrete sampling



**Fig. 5.5** Block scheme of the NRMPC controller in Simulink, for the active vibration cancellation application

time depends on the application. The NRMPC block provides voltage output directly, therefore have to be compensated by the amplifier gain.

#### 5.2.4.9 Custom C functions

In addition to the main C code and BLAS functions, there are two additional custom C functions. One performs element wise division of two vectors, while the other one is an algorithm-specific operation, scaling the elements of a vector by a scalar value and subtracting it from 1. Both of these functions have to be declared similarly to the BLAS routines as externals. Function declarations are as follows:

```
extern double ddot_(int *,double *,int *,double *,int *);
extern void dediv_(int *, double *, double *, double *);
```

Element-wise division of two vectors is performed by the following function:

```
void dediv_(int *n,double *x,double *y,double *z)
{
    int i = 0;
    for (i; i < *n; i++)
    {
        z[i]=x[i]/y[i];
    }
}
```

where vectors  $x$  and  $y$  have a common dimension of  $n$ . Their elements are indexed with the locally declared variable  $i$ . The function takes the dimension  $n$  and vectors  $x$  and  $y$  as its input, and places the result in vector  $z$ .

After the unknown  $\lambda$  is calculated by the algorithm, each element of vector  $x$  is scaled by it. This simple scaling operation is extended by an additional step for computational efficiency. Each scaled element of  $x$  is subtracted from 1, and the result is placed in the vector  $y$ :

```
void descals_(int *n,double *lamN,double *x,double *y)
{
    int i = 0;
    for (i; i < *n; i++)
    {
        y[i]=1-*lamN*x[i];
    }
}
```

The function takes scalar dimension  $n$ , vector  $x$  and scalar  $\lambda$  as its input. The output is placed in  $y$ , where the dimensions of both vectors are  $n$ . An inside loop performs the formerly described simple operation, where the elements are indexed with the locally defined variables  $i$ .

#### 5.2.4.10 The On-line NRMPC Code in C language

A pre-requisite for the correct functionality of this code is the correct calling of external BLAS functions and the xPC optimized BLAS code library. The two external custom functions described in 5.2.4.9 are also needed to be present and properly declared at compilation time.

*At the on-line control process, the following real-time NRMPC algorithm is called on and evaluated at each sampling interval:*

Local variables are declared at the beginning of the code. The BLAS functions require character variables, where for example transposition of matrices is controlled by N and T - as in not to transpose and transpose. Some of these functions also require to mark, whether the upper or lower triangular portion of a symmetric matrix is to be read in.

The order of the system is declared, just as some common values as zero, one or minus one. The value of  $\lambda$  is set to zero at starting time, tolerance and error thresholds are also stated. Finally local matrix and vector variables are declared as well:

```
char *chn="N",*cht="T", *chu="U", *chl="L";
int onei=1, order=2;
double one=1.0, mone=-1.0, zero=0.0, lamN=0,...
...tol=1e-5, err=2e-5;
```

```
double W0, W2, fval, fderval;
double tempv[2], tempm[4], vec[2], tempv2[2],...
... W1[2], W1d[2], W1dd[2], m[2], f[2];
```

After the local variable declarations are expressed, the following mathematical operation is performed in two steps:

$$W_0 = x_0^T Q_{11} x_0 - 1 \quad (5.11)$$

where  $x_0$  marks the current observed state, and  $Q_{11}$  is a partition of the matrix defining the invariant ellipsoid, as calculated in the off-line process.  $W_0$  is a by-product, resulting the logical simplification of matrix operations. The first code line creates a temporary vector, a result of the matrix-vector operation  $Q_{11}x_0$  while the second finishes the task by evaluating the rest:

```
dsymv_(chu,&order,&one,Q11,&order,...
...x0,&onei,&zero,tempv2,&onei);
W0 = ddot_(&order,x0,&onei,tempv2,&onei)-1;
```

In case the resulting vector will be  $W_0 \leq 0$ , the following code portion calculates the next by-product, a vector re-used in later code portions:

$$v = -(R^T Q_{21} x_0) ./ S \quad (5.12)$$

where  $v$  denotes the vector result of this operation, and  $./$  is an element-wise division. This operation is carried out in two steps. First, a general matrix-matrix multiplication saves the result of  $R^T Q_{21}$  into a temporary matrix. Then the expression  $v$  is calculated by multiplying the result with the negative of the current state measurement, and its elements divided by vector  $S$ :

```
if(W0>=0){
dgemm_(cht,chn,&order,&order,&order,&one,...
...R,&order,Q21,&order,&zero,tempm,&order);
dgemv_(chn,&order,&order,&mone,tempm,&order,
...x0,&onei,&zero,W1,&onei);
dediv_(&order,W1,Sm,vec);
```

Another partial result is calculated, by evaluating a dot product of two vectors and adding  $W_0$  to the result:

```
W2 = -ddot_(&order,vec,&onei,W1,&onei)+W0;
```

where in case  $W_2 \geq 0$ , the perturbation vector can be directly calculated by evaluating:

$$f = -Rv \quad (5.13)$$

where  $f$  is the perturbation vector,  $R$  is an input from the off-line optimization, and  $v$  is a vector product re-calculated at each sampling interval.

```

if( W2 >= tol )
  {dgemv_(chn,&order,&order,&mone,R,&order,...
    ...vec,&onei,&zero,f,&onei);}

```

The other option in the *else* construct is to evaluate for the unknown  $\lambda$  using the Newton-Raphson procedure. This conditional statement launches a *while* loop, which cycles through the NR procedure, until the floating point absolute value of error is larger than the pre-set tolerance.

The first part of this code segment serves only to evaluate the matrices used in the NR loop. These simplifications increase computational speed and are based on the assumptions about function  $\Phi(\lambda)$  and its  $i$ -th derivatives.

The second part of the following code segment is the Newton-Raphson algorithm itself. Here the first step is to evaluate the value of  $\Phi(\lambda)$  and its derivative. The ratio of the function value and its derivative is the error, which is subtracted from the result for  $\lambda$  from the previous step:

```

else{
  while(fabs(err)>=tol){
    descals_(&order,&lamN,Sm,m);
    dediv_(&order,W1,m,W1d);
    dediv_(&order,W1d,m,W1dd);
    fval=ddot_(&order,vec,&onei,W1dd,&onei)+W2;
    fderval=2*ddot_(&order,W1d,&onei,W1dd,&onei);
    err=fval/fderval;
    lamN=lamN-err;}

```

Since the value of  $\lambda$  has been acquired in the previous step, the only task left is to evaluate for the perturbation vector  $f$ , which in this case can be also stated as:

$$f = -\lambda TRW_{1d} \quad (5.14)$$

This single mathematical operation is divided into three parts for the C code. First the value of  $\lambda$  is negated, then a temporary vector is created from the product of  $v_{temp} - \lambda RW_{1d}$ . The final step is to calculate  $f$  by multiplying this temporary vector by  $T$  from the left,  $f = Tv_{temp}$ :

```

lamN=-lamN;
dgemv_(chn,&order,&order,&lamN,R,...
  ...&order,W1d,&onei,&zero,&tempv,&onei);
dsymv_(chu,&order,&one,T,&order,...
  ...tempv,&onei,&zero,f,&onei);}

```

With the perturbation value calculated in the previous step, the final task is only to evaluate the current control move according to  $u = Kx_0 + C_0f$ . This is performed in the C code by summing up results of two vector dot operations:

```

u[0] = ddot_(&order,K,&onei,x0,&onei)+

```



```
+ddot_(&order,C0,&onei,f,&onei);}
```

The other option implies that the loop is already optimal, thus the perturbation  $f = 0$ . There is no need for optimization, this is part of an “if - than - else” decision. In this case the fixed feedback matrix is used to calculate the control move from the observed state by evaluating  $u = Kx_0$ . This is again a simple vector dot product:

```
else{
  u[0] = ddot_(&order,K,&onei,x0,&onei);}
```

## 5.2.5 Multi-Parametric MPC

### 5.2.5.1 Introduction

This subsection introduces the use of multi-parametric programming based MPC in active vibration attenuation.

The Multi-Parametric Toolbox (MPT) is a freely available and distributed Matlab toolbox for the design, analysis and rapid deployment of PWA controllers [Kvasnica et al \(2004, 2006\)](#). The multi-parametric control laws created via the MPT toolbox are not only usable in Matlab, but it is possible to deploy them onto rapid software prototyping platforms using the Real-Time Workshop. More details about the MPT toolbox can be found in Chapter 4.

This workbook assumes the current release (Version 2.6.2) of the toolbox, available online<sup>1</sup>. Matlab is assumed to be used for multi-parametric controller calculation and simulations, while the real time code is assumed to be implemented using the xPC Target protocol and Simulink.

### 5.2.5.2 Off-line Controller Computation

The control objective assumed in this example is to regulate toward origin, since the beam equilibrium is located at the origin of the state space. Let the cost function to be set as a quadratic (2-norm). Penalization and constraints are identical to the NRMPC and QPMPC case: input penalty  $R = 10E^{-4}$ , state penalty matrix was set to  $Q = C^T C$ . Input constraints are set to  $\pm 120$  V and output or state constraints are not engaged.

We begin with loading the system model:

```
load n4s2A.mat
sysStruct.A = n4s2.A;
sysStruct.B = n4s2.B;
```

---

<sup>1</sup> Software package and extended documentation is available at:  
<http://control.ee.ethz.ch/~mpt/>

```
sysStruct.C = n4s2.C;
sysStruct.D = 0;
```

The process is followed by naming the state variables and setting constraints on the inputs. Output constraints are set to infinity, therefore practically they are neglected:

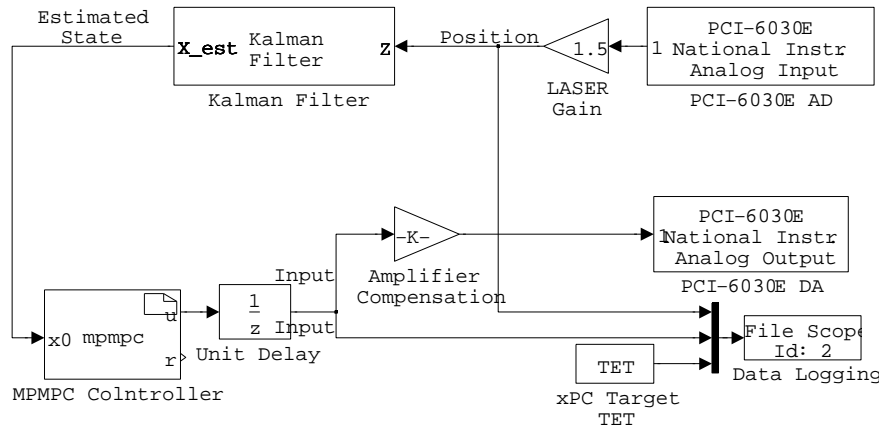
```
sysStruct.StateName{1} = 'x1';
sysStruct.StateName{2} = 'x2';
sysStruct.umin = -120;
sysStruct.umax = 120;
sysStruct.ymin = -inf;
sysStruct.ymax = inf;
```

Sub-optimality level is zero, this means that the software generates an optimal controller with a quadratic cost function - the norm is set to 2. Input and state penalties are defined as well along with the controller horizon:

```
probStruct.norm=2;
probStruct.subopt_lev=0;

probStruct.Q=sysStruct.C'*sysStruct.C;
probStruct.R=1e-4;

probStruct.N=70;
```



**Fig. 5.6** Block scheme of the MPMPC controller in Simulink, for the active vibration cancellation application

The next step is the most important of all, that is the calculation of the controller regions. The main calling function takes the system and the problem structure as an argument and outputs the multi-parametric MPC problem:

```
ctrllex=mpt_control(sysStruct,probStruct);
```

Regions of the controller can be determined by:

```
regions=length(ctrllex);
```

The controller can be saved in its original structure, so it can be later loaded into the workspace or used in off-line simulations via the standard Matlab interface:

```
save ctrllex.mat ctrllex;
```

The export of the controller into a C code is very simple and straightforward, and it can be carried out using the following command:

```
mpt_exportc(ctrllex);
```

One may need such data as the volume of the region of attraction. This for example may be used to compare different controller designs or models. The total volume of the region of attraction is the sum of the individual volumes, and can be simply calculated by:

```
result.areareach=sum(volume(ctrllex.Pn))
```

The maximal absolute deflection of the beam can be calculated by creating a convex hull around the region of attraction, transforming this into a vertex representation and by multiplying the individual edges of the region of attraction with the output matrix  $C$  we may get the direct output equivalents. The maximum of this is the maximal possible deflection at the beam tip:

```
[P,Pn]=hull(ctrllex.Pn);
result.V=extreme(P);
result.maxdef=max(abs(sysStruct.C*result.V'));
```

### 5.2.5.3 On-line Controller Computation

MPT Toolbox features rapid code deployment functionality. The controller stored as a Matlab multi-field variable and can be exported as a stand alone C code by using the command `mpt_exportc(ctrl)`, where `ctrl` is the controller name. This C code can be then integrated into a given application.

The MPMPC controller has been integrated into a custom S-Function block. The function code built through the S-Function Builder takes the state vector as its input and has a single output, the direct controller voltage. The core code is very simple, and involves calling the routine supplied by the MPT Toolbox in the form:

```
double region;
region = mpt_getInput(x0,u);
```

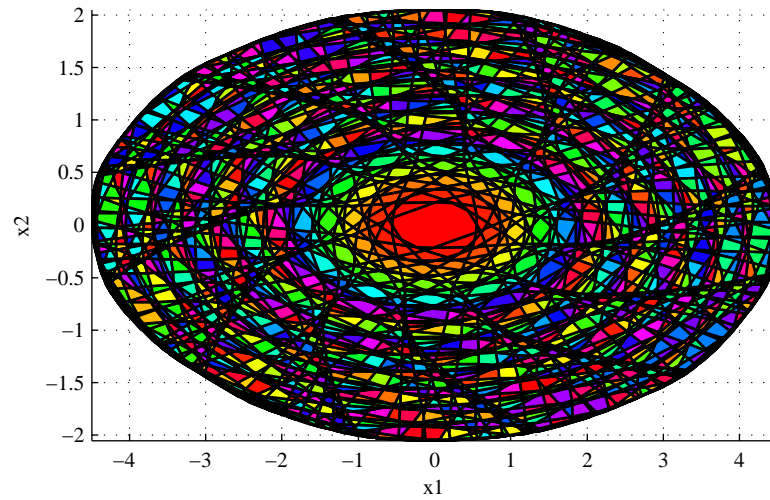


Fig. 5.7 MPMPC controller regions plotted in two dimensional state-space.

where  $x_0$  is the state vector obtained through an observer block,  $u$  is the controller output. An additional variable *region* returns the index number of the polyhedral region corresponding to the acquired current state measurement.

For the correct functionality of the S-Function block the sampling rate is defined, so is the external function declaration of the MPMPC routine:

```
extern double mpt_getInput(double *,double *);
```

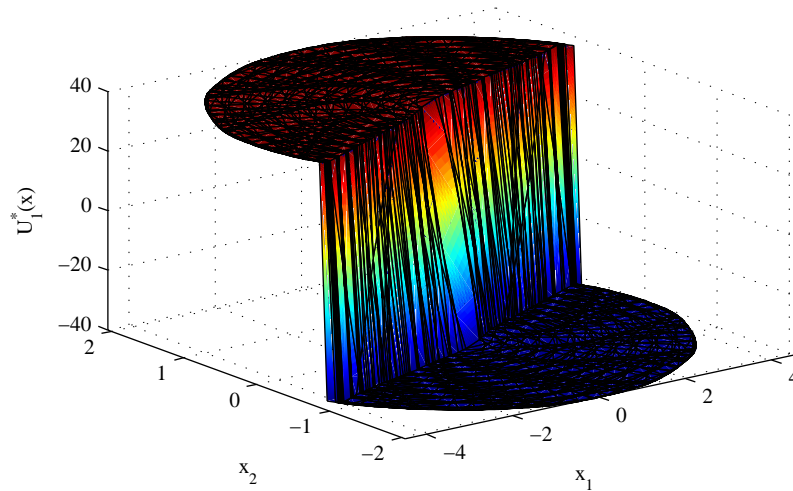
and a declaration of the C source file as well:

```
mpt_getInput.c
```

The process of associating a state with a region and the corresponding PWL function is repeated at each sampling interval and is implemented in the source file *mpt\_getInput.c*, a part of the MPT Toolbox. The controller look-up tables are included in a separate file referenced in this work as *mpt\_getInput.h* and declared within the function source itself.

The block scheme implementing the MPMPC controller loop along with data acquisition, D/A, A/D conversion and other functional parts is featured in Fig. 5.6.

Fig. 5.7 shows multi-parametric MPC controller regions plotted in a two dimensional state-space. Controller partitions shown on the image belong to a controller based on the second order model of the experimental device and conforming to the example presented here. Prediction horizon here is 70 steps. The region of attraction is divided into 10099 polyhedral regions with associated control laws. The figure shows partitions without additional optimization or merging.



**Fig. 5.8** MPMPC controller input in volts.

MPMPC controller action expressed in Volts is plotted against the state space in two dimensions in Fig. 5.8. Note how the control law is essentially reduced to a switching behaviour. In fact many real and simple vibration control systems utilize this highly simplified saturated effect.

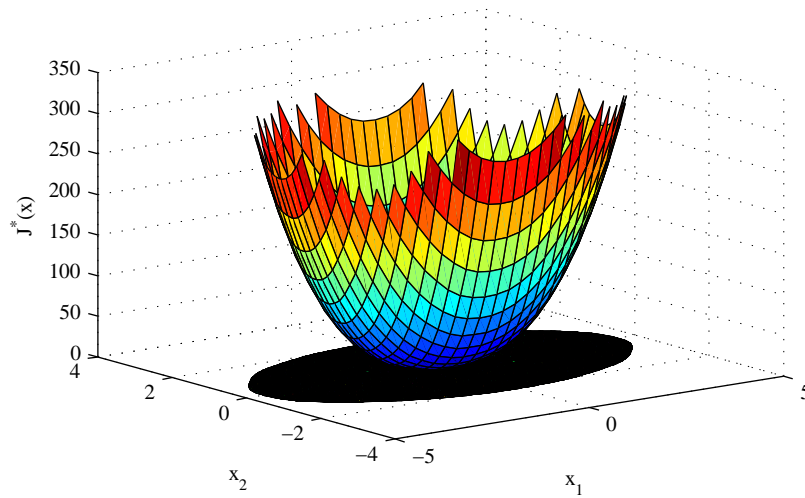
Fig. 5.9 shows the plot of the quadratic cost function value  $J^*$ , related to the polytopic controller regions in state-space.

### 5.2.6 Conclusion

This section has been dealing with the implementation of model predictive control algorithms in the engineering field of active vibration attenuation. Due to the nature of the problem very short sampling times are required, moreover large actuator asymmetry can render the MPC problem quite formidable.

If the control process is augmented with guaranteed stability and feasibility requirements, the useful states are limited inside a portion of the otherwise infinitely large state-space. This portion of the state-space is called the region of attraction. If the expected deformations of under-damped systems belong to a wide scale region, we have to ensure a matching region of attraction.

A good example of such and similar vibrational systems are for example helicopter rotor beams or solar panels in a microgravity environment. This chapter introduced a simple laboratory system, which emulated the physical properties and dynamic behaviour of the aforementioned class of engineering



**Fig. 5.9** MPMPC cost function  $J^*(x)$ .

problems. Details have been given on the hardware implementation of this laboratory test bench, so the if interested one may replicate the setup.

This has been followed by the introduction of three different MPC implementations: traditional dual-mode infinite horizon quadratic programming based MPC, pre-computed explicit multi-parametric programming based MPC and finally a sub-optimal method called Newton-Raphson's MPC. All of these methods are computationally efficient in some way or other: either by design or by implementation details.

A thorough detail is given on all three methods. Although their implementation is connected with the vibration detail, this section may be used as a guide to create a functional MPC controller for any application. Not only the on-line part of the algorithms is discussed in detail, but the off-line part as well. The approach utilized in this chapter is down to Earth and simple, lacking any sort of over-complicated theoretical discussions. Moreover (especially in the case of the QP and MP based implementations) if possible off the shelf and freely available solutions have been preferred.

## 5.3 Implementation of Predictive Control of Air-Fuel Ratio in Spark Ignition Engine

### 5.3.1 Introduction

A run of a spark ignition engine (SI) is highly dependent on the mixture of the sucked air and injected fuel present in the cylinder, waiting to be ignited by the spark. Incorrect ratio of this two components may lead to the poor engine power, ineffective functionality of the catalytic converter resulting in higher level of emissions polluting the environment and in the extreme case this can lead to the engine stoppage. Due to this reason it is crucial to keep the air/fuel ratio (AFR) at the stoichiometric level, which means, that both, the air and the fuel are completely combusted. Due to above mentioned reasons and all the time tightening emission standards the car producers are improving the control of the air/fuel ratio.

Traditional control of air/fuel ratio is based on a feedforward control using predefined tables determining how much fuel has to be injected into a cylinder, based on the information from the mass air flow meter. This fuel amount is subsequently corrected using the information from the lambda probe, so the stoichiometric mixture can be reached. Due to a lambda probe position (at the engine exhaust) a delay arises, causing an improper feedback correction at the unstable engine regimes, like acceleration, or deceleration. On the other side, this kind of control guarantees stability and robustness at all conditions and therefore is still preferred by car producers, despite its disadvantages in control.

The academic field have started to publish other kinds of air/fuel control, mostly model-based ones. The model-based approaches are bringing good quality of control, but are also more sensitive to the model precision and issues with stability and robustness appear. A survey through popular "mean value engine modeling" is described in [Bengtsson et al \(2007\)](#). This analytical way of engine modeling is very clear, but requires exact knowledge of the system and the model error has to be taken into account explicitly. Other ways of a model acquisition are based on the experimental identification (black box modeling). Works of [Zhai et al \(2010\)](#), [Zhai and Yu \(2009\)](#) and [Hou \(2007\)](#) are specialized in employment of neural networks, while [Mao et al \(2009\)](#) uses for engine modeling CARIMA models.

In the engine control itself became popular fuzzy logic ([Hou \(2007\)](#)), neural network control ([Arsie et al \(2008\)](#)) and model predictive control (MPC) approaches ([Lorini et al \(2006\)](#) and [Muske and Jones \(2006\)](#)). General topics on an issue of stability and robustness in MPC can be found in [Mayne et al \(2000b\)](#), or [Zeman and Rohal-Ilkiv \(2003\)](#).

Our approach, introduced in [Polóni et al \(2007\)](#) is utilizing an analytical model predictive controller with a penalization of a terminal state. It uses a multi-model approach using a weighted net (sugeno-type fuzzy logic) of

autoregressive models (ARX) as a system model. The ARX models were identified in the particular working points of the engine as black box models. This method of engine modeling offers an easy way of "global nonlinear system model" acquisition with subsequent utilization in the model based system control. The preliminary real-time predictive control results presented in this paper indicate that the proposed controller could be suitable alternative toward the air/fuel ratio control through the look-up tables.

### 5.3.2 Hardware Description

The engine test bench consists of several components, building up together a flexible and freely programmable system, perfectly suitable for engine control and research.

One part of the test bench is a combustion engine itself, rigidly connected to the eddy current brake. As an interface and a communication part is utilized a rapid control prototyping system based on *dSpace* hardware. This allows to execute the designed controllers of the air/fuel ratio on the engine, instead of the original electronic control unit (ECU).

#### 5.3.2.1 Combustion Engine

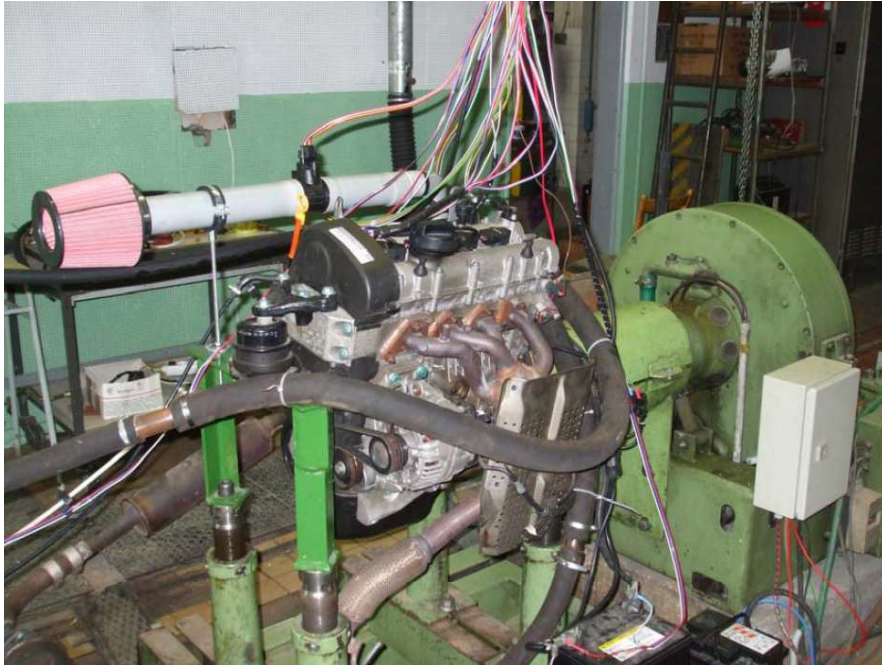
For the experiments and measurements has been used a spark ignition engine from the Volkswagen Polo 1.4 16V (Fig. 5.10). It is a four-cylinder in-line engine with a DOHC valve train.

Engine characteristics:

- Engine code: AUA
- Volume: 1390ccm
- Number of cylinders: 4
- Number of valves: 16
- Bore x stroke: 76.5 x 75.6 mm
- Compression ratio: 10.5
- Power: 55kW (75k) at 5000 rpm
- Torque: 126Nm at 3800 rpm
- Injection system: multipoint, sequential, 1 injector per valve
- Ignition: electronic, independent injectors
- Fuel: Natural 95
- Emissions processing: three way catalys

The original *Bosch* ECU has been completely replaced by a rapid prototyping system. It is dedicated for the control of power stages, as injectors and ignitors, as described in the Subsection 5.3.2.3.





**Fig. 5.10** Spark ignition engine VW Polo 1.4 with a brake

### 5.3.2.2 Engine Brake (Dynamometer)

For the sake of identification experiments resulting in acquisition of the local ARX models of the air and fuel path it was necessary to keep the engine at predefined working points. This aim has been reached by utilizing a *Schenck* dynamometer (Fig. 5.11), loading the engine and so keeping it at the desired revolutions.

The operation of a used brake is based on an eddy current principle. The braking torque is excited by the electromagnetic induction of a coil, which excites eddy currents in the brake's rotor. The brake supports predefined braking profiles, and also manual control of the braking moment. The braking moment is measured by a tensometric force sensor fixed to an arm.

Basic brake features:

- maximal revolutions: 10 000 rpm covered by a voltage output: 0 - 10 V
- maximal braking torque: 750 Nm covered by a voltage output: 0 - 10 V

The control unit of a brake is an independent device allowing the manual brake control by analog voltage signals. It includes the following circuits:

- “thyristor phase” control circuit (braking moment)
- revolutions measurement circuit
- braking moment measurement circuit

The “thyristor phase” control circuit is based on the integrated circuit (IC) TCA785. Function of this IC is a phase control of a brake’s power stage. The control voltage span is 0 - 10 V, corresponding to the braking moment. The braking moment is used for the aim of:

- regulation to constant revolutions, based on a discrete PI controller
- regulation to a particular braking moment

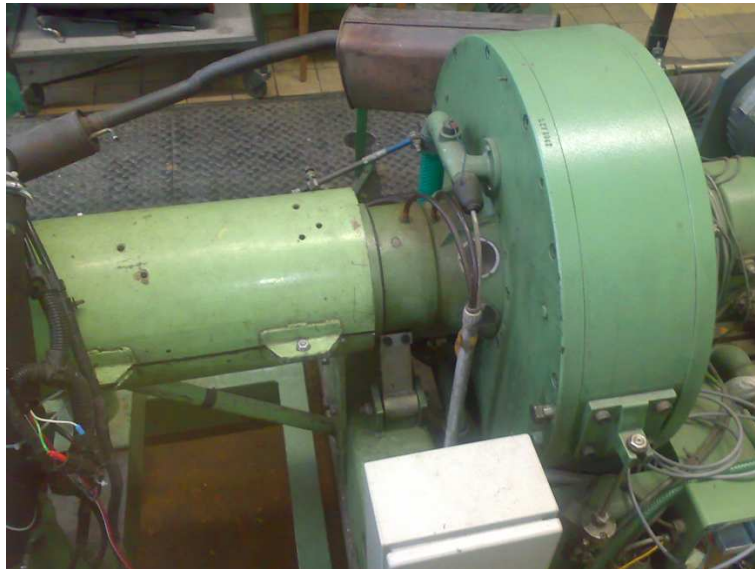


Fig. 5.11 Engine brake

### 5.3.2.3 Rapid Control Prototyping System

The computational unit necessary for the real-time implementation of the MPC control is based on a powerful and freely programmable control system based on *dSpace* and *RapidPro* units; or “Rapid Control Prototyping System” (RCP), (Fig. 5.13, [dSPACE GmbH. \(2009\)](#)).

It is built up on the processor board *ds1005* and hardware-in-loop platform

*dS2202 HIL.*

The main components are (Fig. 5.12):

- DS1005 PPC Board - processor module
- DS2202 HIL I/O Board - module of input/output interface
- DS4121 ECU Interface Board - interface for the communication with the ECU

***DS1005 PPC Board***

DS1005 PPC Board is a core processor module of the *dSpace* system. It has a huge computational power for the real-time applications and creates an interface between the input/output circuits and the host computer. DS1005 is suitable for the systems with fast dynamics with a high computational load.

Basic features:

- processor PowerPC 750GX, 1 GHz
- fully programmable from Simulink environment
- fast communication between dSpace modules over PHS bus

***Module DS2202 HIL I/O Board***

The DS2202 HIL I/O Board module has been designed for the hardware-in-the-loop simulations in automobile industry applications. It supports measurements and control of various peripheral devices typical for different automobile systems.

Module features:

- 20 D/A channels and 16 A/D channels
- 38 digital inputs and 16 digital outputs
- supports 12 to 42V voltage systems

***Module DS4121***

Module DS4121 is an interface between the ECU or RapidPro unit and the dSpace modular system. It has a support for two independent ECUs. The flexibility of this solution allows to control the power stages of the engine, gearbox, or the valve train for a up to 12 cylinder engine.

Module features:

- supports the communication with 8-, 16- and 32-bit architecture of micro controllers
- has 2 LVDS channels for fast communication
- includes interfaces for *MATLAB/Simulink*



**Fig. 5.12** Modules: DS1005; DS2202; DS4121

The RCP ensures sufficient headroom for the real-time execution of complex algorithms (Arsie et al (2008)) and lets all engine tasks to be controlled directly. Also, the customized variants of the controller can be performed immediately.

Typical RCP system consists of:

- A math modeling program (prepared in Simulink)
- Symbolic input/output blocks
- A real-time target computer (embedded computer with an analog and digital I/O)
- A host PC with communication links to target computer
- A graphical user interface (GUI) which enables to control the real time process

The RCP system enables to use a support in the form of embedded functions which make the preparation of algorithms easy and fast. It is a great help, because one can then concentrate on significant problems (development and debugging of algorithms) without the spending time on not so important tasks (how to handle features of RCP system at low level programming).

The RCP is equipped with many input and output ports implemented on the terminal board (Fig. 5.14), managing the whole communication, between the main computational unit and the engine.

To the *RapidPro* is directly connected the knock sensor and the broadband lambda probe. The crankshaft and camshaft sensors are attached through a converter case, developed during the project lifetime. It is utilized by air temperature and pressure sensor and the cooling water and oil temperature sensor, as well. Among other functions it indicates correct functionality of injectors, igniters and oil pressure sensor. Actual battery voltage is displayed. The security circuits protect the engine by switching off the power stages and the fuel pump, if an error occurs.

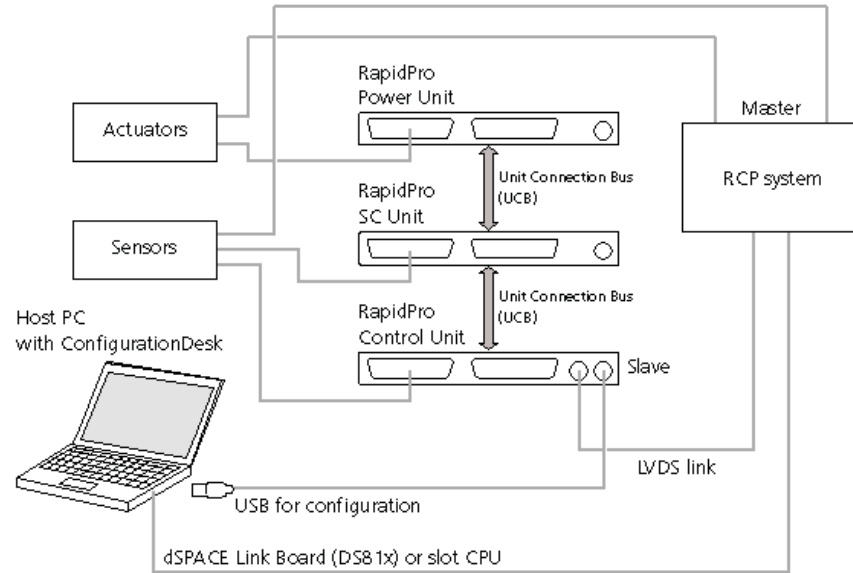


Fig. 5.13 Rapid control prototyping scheme

### 5.3.3 AFR Model Design

#### 5.3.3.1 Air/Fuel Ratio

The model of the air/fuel ratio dynamics  $\lambda$  of a spark ignition engine is based on the mixture, defined as mass ratio of the air and fuel in a time step  $k$ . Due to the fact, that the air mass flow is measured as an absolute value, it was necessary to integrate this amount during the particular time and express the air and fuel quantity as relative mass densities (*grams/cylinder*). Hence, the air/fuel ratio is defined, as:

$$\lambda(k) = \frac{m_a(k)}{L_{th}m_f(k)} \quad (5.15)$$

Where  $m_a(k)$  and  $m_f(k)$  are relative mass amounts of air and fuel in a cylinder and  $L_{th} \approx 14.64$  is the theoretical amount of air necessary for the ideal combustion of a unit amount of fuel.

Considering the  $\lambda(k)$  modeling, the engine has been divided into two subsystems with independent inputs, namely into:

- air path* with the air throttle position as the disturbance input, and
- fuel path* with the input of fuel injector opening time.

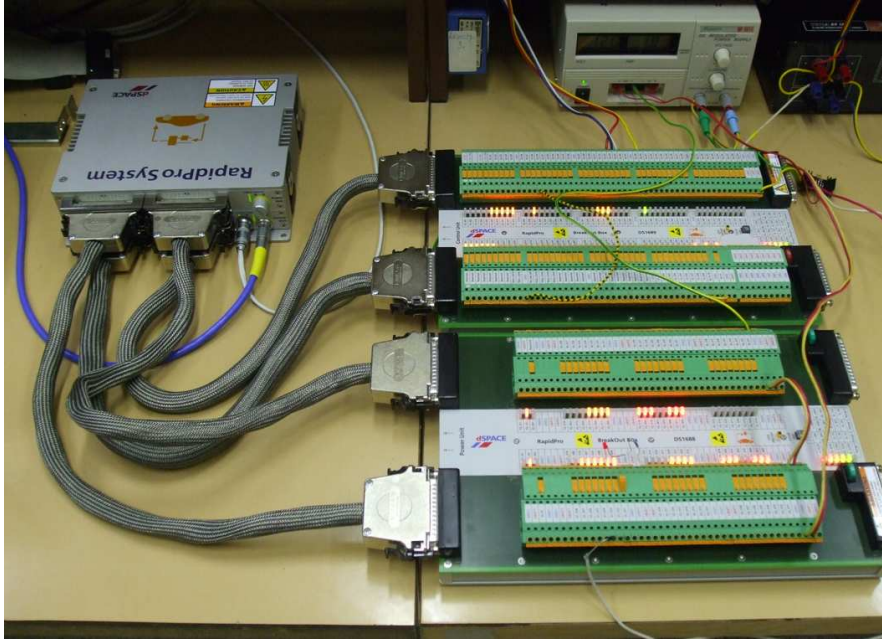


Fig. 5.14 RapidPro unit with a terminal board



Fig. 5.15 Case with converters and security circuits

Another disturbance-like acting quantity were engine revolutions, implicitly included in the engine model, particularly for each working point. The output ratio of both paths is the value of  $\lambda$ .

### 5.3.3.2 SI Engine Modeling using ARX Models

The engine modeling is based on the weighted linear local model with single input single output (SISO) structure (Polóni et al (2008)). The parameters of local linear ARX models with weighted validity (Murray-Smith and Johanssen (1997)) are identified to model AFR nonlinear dynamics. The principle of this nonlinear modeling technique is in partitioning of the engine's working range into smaller working points.

A net of local ARX models weighted for a particular working point  $\phi$  is defined, as:

$$\sum_{h=1}^{n_M} \rho_h(\phi(k)) A_h(q) y(k) = \sum_{h=1}^{n_M} \rho_h(\phi(k)) B_h(q) u(k) + \sum_{h=1}^{n_M} \rho_h(\phi(k)) c_h + e(k) \quad (5.16)$$

defined by polynomials  $A_h$  and  $B_h$ :

$$\begin{aligned} A_h(q) &= 1 + a_{h,1}q^{-1} + \dots + a_{h,n_y}q^{-n_y} \\ B_h(q) &= b_{h,1+d_h}q^{-1-d_h} + \dots + b_{h,n_u+d_h}q^{-n_u-d_h} \end{aligned} \quad (5.17)$$

where symbolics  $q^{-i}$  denotes a sample delay, e.x.  $q^{-i}y(k) = y(k-i)$ ,  $a_{h,i}$  and  $b_{h,(j+d_h)}$  are parameters of  $h$ -th local function and  $d_h$  is its delay. Parameter  $n_M$  represents the number of local models.

The  $\rho_h$  denotes a weighting function for a particular ARX model (see Sec. 5.3.3.4) and  $e(k)$  is a stochastic term with a white noise properties. The engine working point itself is defined by engine revolutions  $n_{en}$  and the throttle valve position  $t_r$ , hence:  $\phi(k) = [n_{en}(k), t_r(k)]^T$ . The absolute term  $\hat{c}_h$  of the equation is computed from the steady state values of the system output  $y_{e,h}$  and the system input  $u_{e,h}$ , as:

$$\hat{c}_h = y_{e,h} + y_{e,h} \sum_{i=1}^{n_y} \hat{a}_{h,i} - u_{e,h} \sum_{j=1}^{n_u} \hat{b}_{h,j} \quad (5.18)$$

The model output is computed from the equation:

$$\begin{aligned} y_s(k) &= \sum_{h=1}^{n_M} \rho_h(\phi(k)) \\ &\cdot \left( \sum_{i=1}^{n_y} \hat{a}_{h,i} q^{-i} y_s(k) + \sum_{j=1}^{n_u} \hat{b}_{h,(j+d_h)} q^{-j-d_h} u(k) + \hat{c}_h \right) \end{aligned} \quad (5.19)$$

Introducing the estimated parameter vector  $\hat{\theta}_h$  and the regression vector  $\gamma(k)$ , equation (5.19) becomes:

$$y_s(k) = \gamma^T(k) \sum_{h=1}^{n_M} \rho_h(\phi(k)) \hat{\theta}_h + \sum_{h=1}^{n_M} \rho_h(\phi(k)) \hat{c}_h \quad (5.20)$$

### 5.3.3.3 Model Identification

Parameters of the local ARX models have been estimated from the data acquired from the exhaust gas oxygen sensor and an air flow sensor. The identification has been designed so, that the dynamics of the air path and fuel path stayed uncoupled. Dynamics of both paths were measured indirectly.

The first experiment started at the stoichiometric value of  $\lambda_a$  in the operation point  $\phi$ . To excite the air path dynamics, the throttle valve position was oscillating around its steady position according to a pseudo-random binary signal (PRBS, Fig. 5.16), while the fuel injectors were delivering constant fuel mass  $m_{f,e}$ . The change in  $\lambda_a$  value has been recorded. During the experiment the engine had been braked at constant revolutions.

The PRBS signal is a signal changing its value binary, that means from one level to another with a pseudo-randomly changing period (Fig. 16(a)). This period has to be chosen with respect to the identified system so, that for even smallest change in PRBS a clear system response can be noticed. This condition has to be ensured, to excite the entire system dynamics and consequently to perform a relevant system identification. The Fig. 16(b) shows an example of a correctly identified system with a good fit, compared to the original data (system output).

The identification of the fuel path dynamics has been done similarly, but with the fixed throttle valve delivering a constant air mass  $m_{a,e}$ . The PRBS is varying the fuel injectors' opening time. In both experiments it was necessary to wisely propose a PRBS, so that the air/fuel mixture is always ignitable.

The local ARX models can be subsequently determined from the measured values of instantaneous  $\lambda_a(k)$  and  $\lambda_f(k)$  belonging to the air path and fuel path, utilizing relative air and fuel mass densities, as:

$$m_a(k) = m_{a,e}(\phi)\lambda_a(k) \quad (5.21)$$

$$m_f(k) = \frac{m_{f,e}(\phi)}{\lambda_f(k)} \quad (5.22)$$

The final formula describing the air/fuel ratio dynamics is built up of local linear ARX models of the air and fuel paths is in the form:

$$\lambda_s(k) = \frac{1}{L_{th}} \cdot \left[ \frac{\gamma_a^T(k) \sum_{h=1}^{n_A} \rho_{a,h}(\phi(k)) \hat{\theta}_{a,h} + \sum_{h=1}^{n_A} \rho_{a,h}(\phi(k)) \hat{e}_{a,h}}{\gamma_f^T(k) \sum_{h=1}^{n_F} \rho_{f,h}(\phi(k)) \hat{\theta}_{f,h} + \sum_{h=1}^{n_F} \rho_{f,h}(\phi(k)) \hat{e}_{f,h}} \right] \quad (5.23)$$

Where:

- $\gamma$  is the regression vector of system inputs and outputs
- $n_A$  is the amount of working points
- $\rho$  is the interpolation function
- $\phi$  is the vector of a working point



$\theta$  is the vector of ARX parameters  
 $c$  is the absolute term of an ARX model

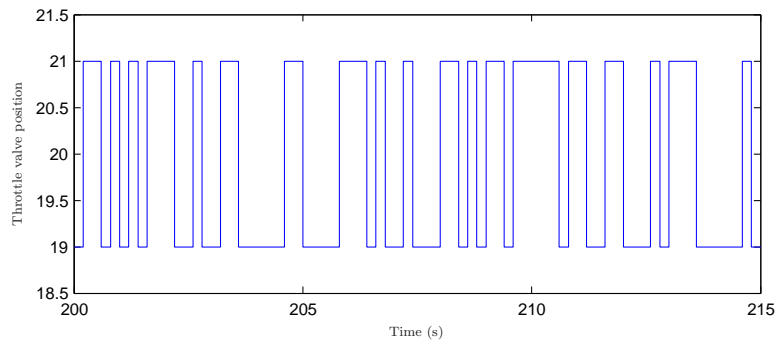
In accordance with the general model structure presented, the key variables used for the air- and fuel path are defined in the Table 5.1 and coupled with the general variables used in the presented formulas.

**Table 5.1** Symbol connection between general expression and the model

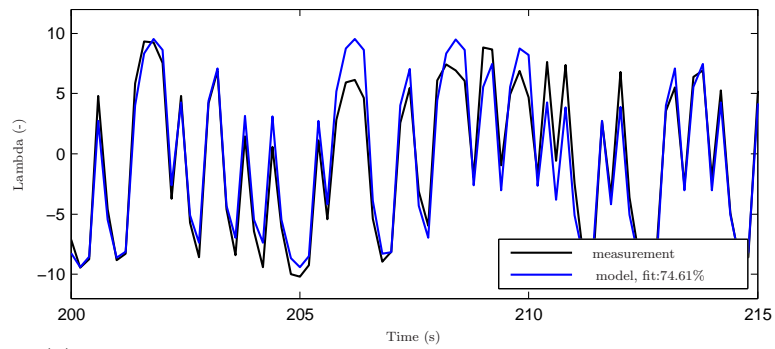
general symbol	air-path model	fuel-path model	operating point
$y(k)$	$m_a(k)$	$m_f(k)$	
$u(k)$	$t_r(k)$	$u_f(k)$	
$\gamma(k)$	$\gamma_a(k)$	$\gamma_f(k)$	
$\hat{\theta}_h$	$\hat{\theta}_{a,h}$	$\hat{\theta}_{f,h}$	
$\rho_h(\phi(k))$	$\rho_{a,h}(\phi(k))$	$\rho_{f,h}(\phi(k))$	
$\hat{c}_h$	$\hat{c}_{a,h}$	$\hat{c}_{f,h}$	
$\phi(k)$			$[n_e(k), t_r(k - \delta)]^T$

The schematics of the engine identification, described above in this Section is depicted in Fig 5.19. The particular identification results of the air- and fuel paths are shown in Fig. 5.17 and Fig. 5.18, respectively. The figures show static characteristics of both engine paths, defining the relative amount of air and fuel flowing through the engine as a function of throttle valve opening angle, or the injector opening time, in the case of fuel injectors; at specific engine revolutions. The step responses of both engine subsystems for various engine revolutions are depicted, as well.

The identified model parameters are situated in Tab. 5.2 and 5.3. These are at the same time the final results of the identification and they used for the prediction of future states of the system in the controller.

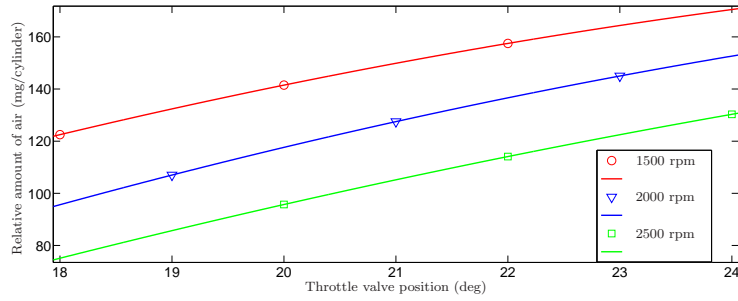


(a) Excitation PRBS signal

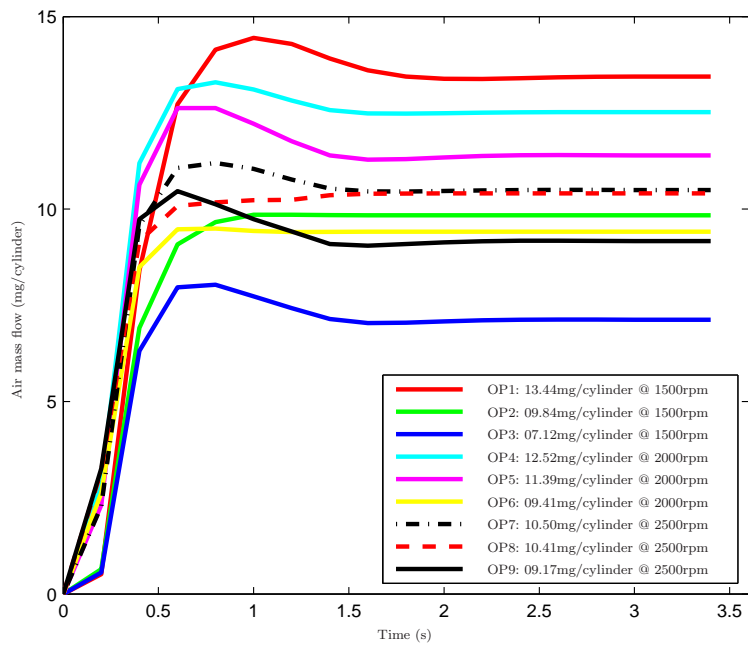


(b) Response curve of the simulated system to the PRBS excitation

**Fig. 5.16** ARX identification at a particular working point (air path)

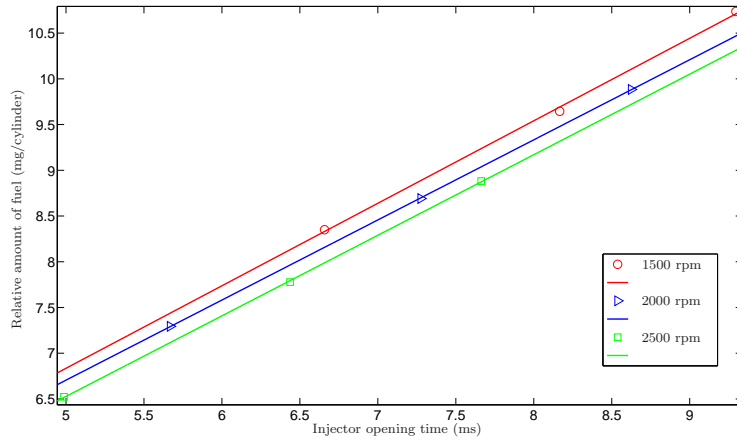


(a) Static characteristics of the air path

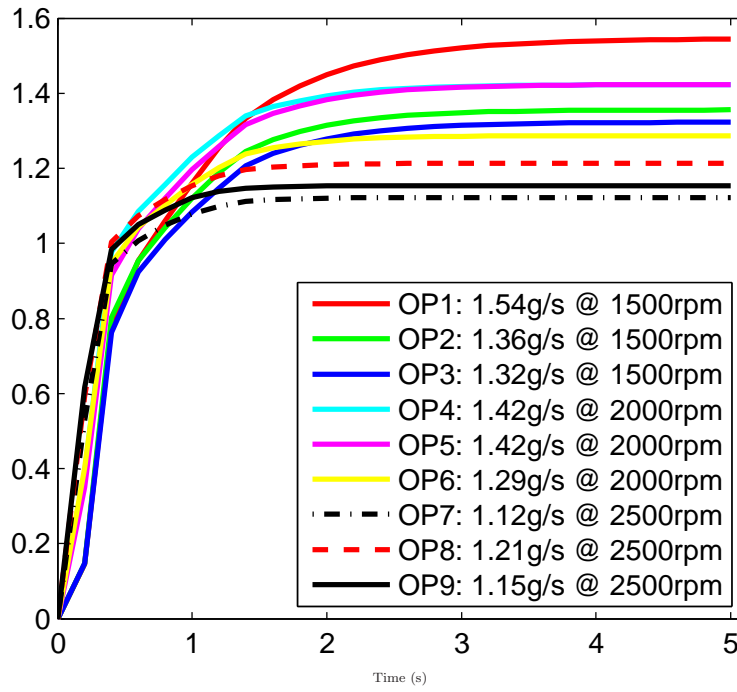


(b) Response curve of the air path in the operating points

Fig. 5.17 Results of the air path identification



(a) Static characteristics of the fuel path



(b) Response curve of the fuel path in the operating points

Fig. 5.18 Results of the fuel path identification

**Table 5.2** Air path model parameters

$h, OP$	$\tilde{\theta}_{a,h} = [\tilde{a}_{h,1} \quad \tilde{a}_{h,2} \quad \tilde{a}_{h,3} \quad \tilde{a}_{h,4} \quad \tilde{a}_{h,5} \quad \tilde{b}_{h,1} \quad \tilde{b}_{h,2}]^T$								$c_h$	$t_r(^{\circ})$	$m_{a,e}(mg/v.)$	$ot.(rpm)$
1	-0.5558	0.1253	-0.0095	0.0235	0.0204	0.5164	7.6000	-72.2300	18	122.5	1500	
2	-0.3492	0.0299	-0.0104	0.0119	-0.0024	0.6424	6.0450	-37.8500	20	141.5	1500	
3	-0.2938	0.0698	0.0331	0.0251	0.0295	0.5599	5.5910	-0.1069	22	157.5	1500	
4	-0.2399	0.0288	0.0148	0.0188	0.0175	2.8360	7.6770	-109.9813	19	107.5	2000	
5	-0.2534	0.0522	0.0292	0.0274	0.0266	2.3090	7.7400	-98.4255	21	127.5	2000	
6	-0.1782	0.0245	0.0054	0.0032	-0.0023	2.6220	5.4040	-60.9862	23	145.0	2000	
7	-0.2118	0.0221	0.0126	0.0247	0.0188	2.2820	6.8110	-98.1203	20	95.7	2500	
8	-0.1633	0.0186	-0.0175	0.0146	-0.0220	3.1980	5.4450	-98.2250	22	114.1	2500	
9	-0.1364	0.0486	0.0383	0.0167	0.0464	3.2560	6.0385	-89.0348	24	130.3	2500	

**Table 5.3** Fuel path model parameters

$h, OP$	$\tilde{\theta}_{a,h} = [\tilde{a}_{h,1} \quad \tilde{a}_{h,2} \quad \tilde{a}_{h,3} \quad \tilde{a}_{h,4} \quad \tilde{a}_{h,5} \quad \tilde{b}_{h,1} \quad \tilde{b}_{h,2}]^T$								$c_h$	$u(ms)$	$m_{f,e}(mg/v.)$	$ot.(rpm)$
1	-0.2516	-0.0827	-0.0771	-0.0636	-0.0431	0.1463	0.5987	-0.9737	6.658	8.2735	1500	
2	-0.2132	-0.0728	-0.0636	-0.0441	-0.0373	0.1474	0.6242	-0.7749	8.166	9.7026	1500	
3	-0.2470	-0.0602	-0.0607	-0.0377	-0.0448	0.1467	0.5805	-0.8746	9.296	10.7074	1500	
4	-0.1571	-0.0402	-0.0835	-0.0181	-0.0579	0.3897	0.5257	-0.4549	5.668	7.3588	2000	
5	-0.1888	-0.0475	-0.0865	-0.0203	-0.0601	0.3509	0.4987	-0.9676	7.275	8.7353	2000	
6	-0.1555	-0.0299	-0.0667	-0.0183	-0.0428	0.4020	0.4820	-0.7991	8.625	9.9395	2000	
7	-0.1217	-0.0218	-0.0471	-0.0077	-0.0168	0.5273	0.3537	0.7585	4.987	6.5651	2500	
8	-0.1715	0.0025	-0.0562	-0.0081	-0.0227	0.5931	0.3097	-0.0154	6.437	7.7898	2500	
9	-0.1935	0.0083	-0.0478	-0.0136	-0.0041	0.6168	0.2477	0.0373	7.663	8.8919	2500	

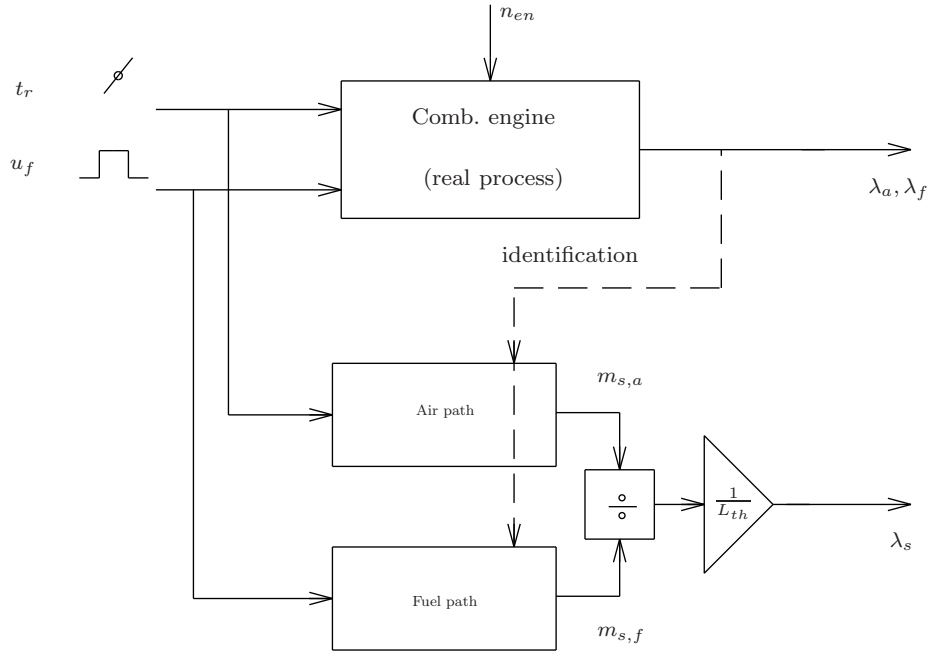


Fig. 5.19 Scheme of the identification process

### 5.3.3.4 Weighting Functions

The full working range of the engine has been covered by discrete amount of local linear models (LLMs), identified at particular working points. Due to a request of a smooth and continuous global engine model, design of weighting functions, defining validity of each local model according to an instantaneous working point of the engine was crucial. There were designed particular interpolation functions for every LLM, assigning it 100% validity exactly at the belonging working point with decreasing trend depending on the change of the throttle valve opening  $t_r$  or the engine revolutions  $n_{en}$ . The Gaussian functions were used as the local weighting functions, due to their suitable shape fulfilling approximation properties. Each one is defined, as:

$$\tilde{\rho}_h(\phi(k)) = \exp \left[ - \left[ \Delta n_{en}(k) \quad \Delta t_r(k) \right] \begin{bmatrix} \frac{1}{\sigma_{h,1}^2} & 0 \\ 0 & \frac{1}{\sigma_{h,2}^2} \end{bmatrix} \begin{bmatrix} \Delta n_{en}(k) \\ \Delta t_r(k) \end{bmatrix} \right] \quad (5.24)$$

The choice of tuning parameters  $\sigma_{h,1} = 250$  and  $\sigma_{h,2} = 0.8$  used in the weighting functions has been chosen experimentally, awaiting continuous and

smooth output of the modeled system. At the same time it has been found out, that there can be used identical weighting functions for weighting of an air and fuel path parameters, as well.

All the weighting functions were at the end normalized (5.25), so the sum of values of all weighting functions belonging to a particular working point (Fig. 5.20), equals exactly one, or:  $\sum_{h=1}^{n_M} \rho_h(\phi(k)) = 1$ .

$$\rho_h(\phi(k)) = \frac{\tilde{\rho}_h(\phi(k))}{\sum_{h=1}^{n_M} \tilde{\rho}_h(\phi(k))} \quad (5.25)$$

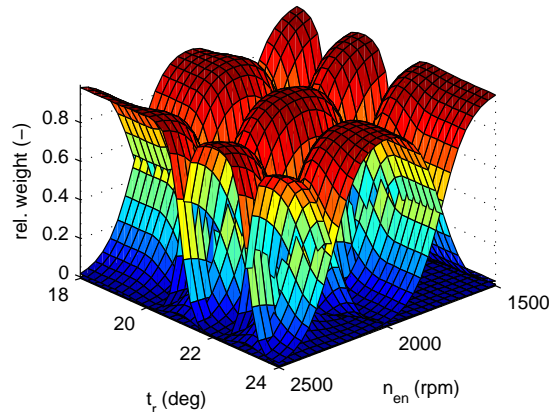


Fig. 5.20 Relative weighting Gaussian functions

### 5.3.4 Predictive Control

The strategy of an “exceeding oxygen amount” control using a predictive controller is based on a prediction of a controlled quantity  $\lambda$  and subsequent minimization of a chosen cost function on the horizon  $N_p$  expressed in a standard quadratic form. The value of  $\lambda$  is predicted by utilization of partially linear models of the air and fuel path. Through the independent air path model the proper amount of fuel is predicted and enters the cost function  $J$ . Hence, the target of the cost function minimization is to determine such a control law, that the measured system output  $\lambda$  is stoichiometric. The second modeled subsystem, the fuel-path, is an explicit component of the objective function where the amount of the fuel is the function of optimized control action (Polóni et al (2008)).

### 5.3.4.1 Predictive Model

The applied control strategy is based on the knowledge of the internal model (IM) of air-path, predicting the change of air flow through the exhaust pipe, and consequently, setting the profile of desired values of the objective function on the control horizon. In this case we will consider the state space (SS) formulation of the system, therefore it is necessary to express linear local ARX models in parameter varying realigned SS model:

$$\begin{aligned} x_{(a,f)}(k+1) &= A_{(a,f)}(\phi)x_{(a,f)}(k) + B_{(a,f)}(\phi)u_{(a,f)}(k) \\ m_{s,(a,f)}(k) &= C_{(a,f)}x_{(a,f)}(k) \end{aligned} \quad (5.26)$$

The weighted parameters of a multi-ARX models are displayed in matrices  $A_{a,f}$  and  $B_{a,f}$  for both subsystems. This is a non-minimal SS representation whose advantage is, that no state observer is needed. The “fuel pulse width control” is tracking the air mass changing on a prediction horizon from IM of the air-path, by changing the amount of injected fuel mass. Due to tracking offset elimination, the SS model of the fuel-path (5.26) (index  $f$ ), with its state space vector  $x_f$ , is written in augmented SS model form to incorporate the integral action

$$\begin{aligned} \tilde{x}_f(k+1) &= \tilde{A}_f(\phi)\tilde{x}_f(k) + \tilde{B}_f(\phi)\Delta u_f(k) \quad (5.27) \\ &\text{or} \\ \begin{bmatrix} x_f(k+1) \\ u_f(k) \end{bmatrix} &= \begin{bmatrix} A_f(\phi) & B_f(\phi) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_f(k) \\ u_f(k-1) \end{bmatrix} + \\ &\quad + \begin{bmatrix} B_f(\phi) \\ 1 \end{bmatrix} \Delta u_f(k) \end{aligned}$$

$$m_{s,f}(k) = \tilde{C}_f\tilde{x}_f(k) + D_f\Delta u_f(k) \quad (5.28)$$

or

$$m_{s,f}(k) = [C_f \ D_f] \tilde{x}_f(k) + D_f\Delta u_f(k)$$

The prediction of the air mass ( $\underline{m}_a$ ) on the prediction horizon ( $N$ ) is solely dependent on the throttle position ( $\underline{t}_r$ ) and is computed as

$$\underline{m}_a(k) = \Gamma_a(\phi)x_a(k) + \Omega_a(\phi)\underline{t}_r(k-1) \quad (5.29)$$

where the  $x_a$  denotes the state space vector of the air path.

Due to the unprecise modeling (IM strategy), biased predictions of the air mass future trajectory and consequently biased fuel mass might occur. This error can be compensated by the term  $L[\hat{m}_f(k) - m_{s,f}(k)]$  in the fuel mass prediction equation ( $\underline{m}_f$ )



$$\begin{aligned} \underline{m}_f(k) = & \Gamma_f(\phi)\tilde{x}_f(k) + \Omega_f(\phi)\Delta\underline{u}_f(k-1) + \\ & + L[\hat{m}_f(k) - m_{s,f}(k)] \end{aligned} \quad (5.30)$$

The matrices of free response  $\Gamma_a$ ,  $\Gamma_f$  and forced response  $\Omega_a$ ,  $\Omega_f$  are computed from models (5.26) and (5.27), respectively (Maciejowski (2000)). Since there is only  $\lambda(k)$  measurable in equation (5.15), the value of  $m_a(k)$  needs to be substituted using IM of the air-path, then

$$\hat{m}_f(k) = \frac{1}{L_{th}} \frac{m_{s,a}(k)}{\lambda(k)} \quad (5.31)$$

The estimate  $\hat{m}_f(k)$  is used to compensate for possible bias errors of predicted  $\underline{m}_f(k)$  in (5.30).

### 5.3.4.2 Cost Function

The main part of a cost function is a sum of deviations of predicted outputs from a set point, i.e. a weight of a future control deviations. Its another elements are a penalization of control increments  $r$ ; and a  $p$  penalizing a deviation between a predicted and desired end state.

To eliminate a the steady state control error, the criterion (5.32) is defined through the control increments. That guarantees in a steady state, that  $J_\lambda = 0$  also for the system with no integration properties, what is the case of the fuel path

$$\begin{aligned} J_\lambda = & \left\| \frac{m_a(k)}{L_{th}} - \underline{m}_f(k) \right\|_2^2 + r \|\Delta\underline{u}_f(k-1)\|_2^2 \\ & + p \|\tilde{x}_f(N) - \tilde{x}_{f,r}(N)\|_2^2 \end{aligned} \quad (5.32)$$

The chosen MPC approach utilizes the state space representation and its structure uses a control deviation for the correction of the prediction.

Due to a disturbance  $d(k)$ , the steady state values of  $u$  and  $x$  have to be adapted so, that the assumption  $J = 0$  is valid. This problem solves an explicit inclusion of the disturbance into the model.

The fuel injectors are controlled by a fuel pulse width, what is at the same time the control  $u_f$ . The optimal injection time can be computed by minimization of a cost function (5.32), which has after expansion by the fuel path prediction equation, form:

$$\begin{aligned} J_\lambda = & \left\| \frac{m_a}{L_{th}} - \Gamma_f\tilde{x}_f(k) + \Omega_f\Delta\underline{u}_f(k-1) + L[\hat{m}_f(k) - m_{s,f}(k)] \right\|_2^2 \\ & + r \|\Delta\underline{u}_f(k-1)\|_2^2 + p \|\tilde{x}_f(N) - \tilde{x}_{f,r}(N)\|_2^2 \end{aligned} \quad (5.33)$$

An analytical solution of  $\frac{dJ_\lambda}{d\underline{u}} = 0$  of (5.33) without constraints leads to a definition determining the change of fuel injector opening time in a step ( $k$ ),

as:

$$\Delta u = (\Omega^T \Omega + Ir + p\Omega_{xN}^T \Omega_{xN})^{-1} \cdot [\Omega^T [w(k) - \Gamma \tilde{x}(k) - L(y(k) - y_s(k))] - p\Omega_{xN}^T A^N \tilde{x}(k) + p\Omega_{xN}^T \tilde{x}_{f,r}(N)] \quad (5.34)$$

Hence, the absolute value of the control action in a step  $k$  is given by a sum of a newly computed increment in a control (5.34) and an absolute value of the control in a step  $(k - 1)$ :

$$u_f(k) = u_f(k - 1) + \Delta u_f(k) \quad (5.35)$$

### 5.3.5 Results of a Real-Time Application of a Predictive Control

The ability to control the mixture concentration at stoichiometric level using MPC is demonstrated through the real-time SI engine control (Fig. 5.21). This has been performed using the AFR predictive control strategy described

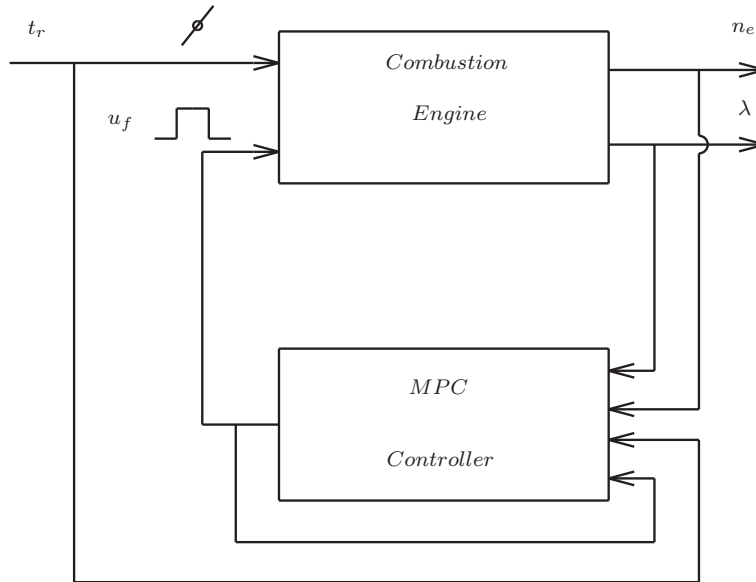
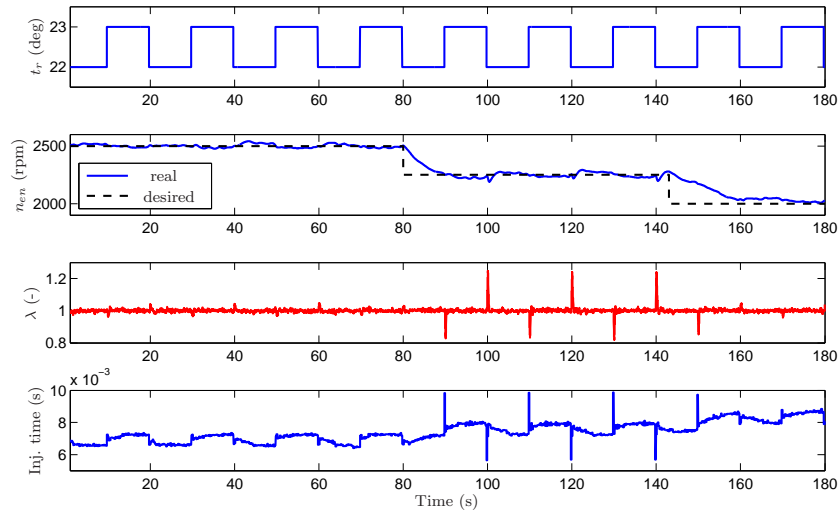


Fig. 5.21 Control scheme

in the previous section, designed in *Matlab/Simulink* environment and compiled as a real-time application for a *dSpace* platform. It has been applied to the VW Polo engine (Fig. 5.10), 1390 cm<sup>3</sup> with 55 kW@5000 rpm, not equipped with a turbocharger or an exhaust gas recirculation system. The

control period was 0.2 s. The result of an identification are 9 local linear models (LLM) for each, air and fuel path, dependent on a throttle valve opening and engine revolutions.

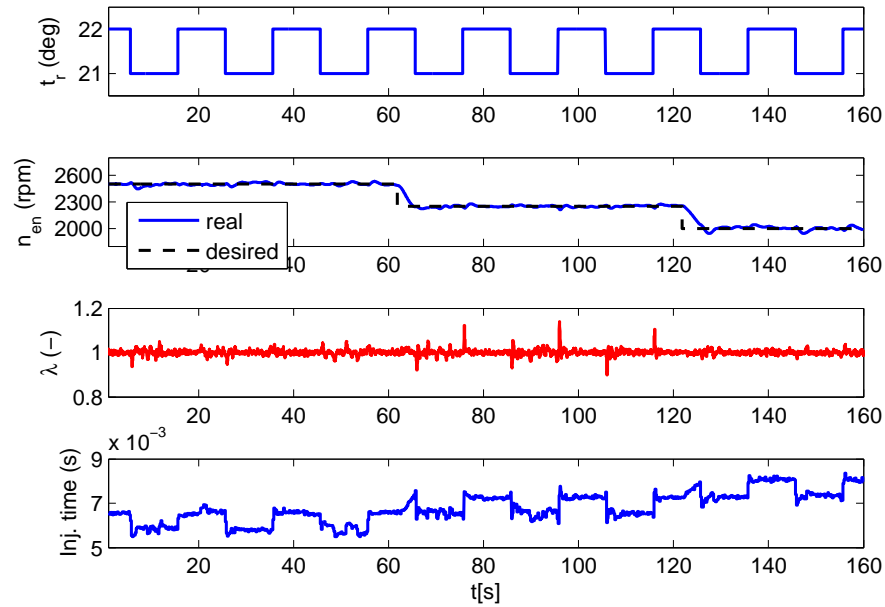


**Fig. 5.22** Results of a real-time control of the SI engine (without “desired final state” penalization)

The primary target of a control (Fig. 5.23) was to hold the air/fuel ratio in a stoichiometric region ( $\lambda = 1$ ), in the worst case to keep the mixture ignitable ( $0.7 \leq \lambda \leq 1.2$ ).

During the experiment, the change in throttle valve opening, between 21 and 22 degrees (Fig. 5.23, variable  $t_r$ ) and the change of engine revolutions (Fig. 5.23, variable  $n_{en}$ ), has been performed, at the same time. These changes simulate varying working regimes of an engine, which is adapting its run to a daily traffic. Changes in  $t_r$  and  $n_{en}$  quantities are determining the engine load, at the same time, ensuring, that the engine passes through several working points during its operation. As mentioned in Section 5.3.3.2, the engine revolutions are not included among explicit variables of local models, but they build together with a delayed throttle valve position a vector of an working point  $\phi(k)$ .

The quality of control is sufficient (Fig. 5.23, variable  $\lambda$ ), with exceptional acceptable overshoots in both directions. These overshoots of the controlled variable  $\lambda$  have been caused by smaller model precision, due to its distance from the working point, at which the system identification has been performed. This effect is caused by the approximation of a particular model



**Fig. 5.23** Results of a real-time control of the SI engine

from the other working points' models.

The overshoots were also diminished by an introduction of a penalization of a “difference from the terminal state”. This means, that into cost function is added a member, comprising the difference, between the instantaneous and the desired terminal state of a state vector. As it can be seen, skipping this member in a cost function reduces the quality of control significantly (Fig. 5.22). The corresponding control action computed by the controller is shown in (Fig. 5.23, variable *Inj.time*).

The initial engine warm-up (to 80 °C ) eliminated model-plant mismatch caused by temperature dependent behavior of the engine.

The control has been performed by choosing the penalization  $r = 0.1$ . Utilizing the member  $p\|\tilde{x}_f(N) - \tilde{x}_{f,r}(N)\|_2^2$  of a cost function by setting  $p = 1.0$  allowed us to shorten the control horizon to  $N_p = 20$  what significantly unloaded the computational unit and stabilized the controlled output of the engine on this shortened horizon, as well. The best control has been achieved in the neighborhood of working points, what is logically connected to the most precise engine model at those points. In other working points the control is still good enough, with small deviations from the stoichiometric mixture.

### 5.3.6 Conclusion

Considering the preliminary results from the real-time experiments at the engine, it can be concluded, that the idea of the AFR model predictive control based on local ARX models is suitable and applicable for the SI engine control. Proposed flexible design of a predictive controller offers easy tuning possibilities, extension of the global engine model to other working regimes of the engine and implementation of constraints. This fact also indicates the next project step, which is the overshoot elimination in the  $\lambda$  - control by the identification of wider net of “local engine models”.

**Acknowledgements** The authors gratefully acknowledge the financial support granted by the Slovak Research and Development Agency under the contracts LPP-0096-07, LPP-0075-09 and LPP-0118-09. This research is also supported by the grant from Iceland, Liechtenstein and Norway through the EEA Financial Mechanism and the Norwegian Financial Mechanism. This grant is co-financed from the state budget of the Slovak Republic.

## References

- Anstee R (2006) The Newton-Rhapson Method. Lecture notes
- Arsie I, Iorio SD, Noschese G, Pianese C, Sorrentino M (2008) Optimal air-fuel ratio. *dSpace Magazine* (1):20–23
- Bengtsson J, Strandh P, Johansson R, Tunestal P, Johansson B (2007) Hybrid modeling of homogenous charge compression ignition (HCCI) engine dynamics -a survey. *International journal of control* 80(11):1814–1847
- Cannon M, Kouvaritakis B (2005) Optimizing Prediction Dynamics for Robust MPC. *IEEE Transactions on Automatic Control* 50(11):1892–1597
- Chen H, Allgöwer F (1998) A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability. *Automatica* 34(10):1205–1217
- Dong X, Meng G, Peng J (2006) Vibration control of piezoelectric smart structures based on system identification technique: Numerical simulation and experimental study. *Journal of Sound and Vibration* 273:680–693
- Ferreau H (2006) An online active set strategy for fast solution of parametric quadratic programs with applications to predictive engine control. Master’s thesis, University of Heidelberg
- Ferreau H, Bock H, Diehl M (2008) An online active set strategy to overcome the limitations of explicit mpc. *International Journal of Robust and Nonlinear Control* 18(8):816–830
- dSPACE GmbH (2009) HelpDesk Application
- Hou Z (2007) Air fuel ratio control for gasoline engine using neural network multi-step predictive model. 3rd international conference on intelligent computing, Qingdao, China
- Kouvaritakis B, Rossiter J, Schuurmans J (2000) Efficient robust predictive control. *IEEE Transactions on Automatic Control* 45(8):1545–1549
- Kouvaritakis B, Cannon M, Rossiter J (2002) Who needs QP for linear MPC anyway? *Automatica* 38:879–884

- Kvasnica M, Grieder P, Baotić M (2004) Multi-Parametric Toolbox (MPT). Online, available: <http://control.ee.ethz.ch/>
- Kvasnica M, Grieder P, Baotic M, Christophersen FJ (2006) Multi-Parametric Toolbox (MPT). Extended documentation
- Lara A, Burch JC, Schloss JM, Sadek IS, Adali S (2003) Piezo patch sensor/actuator control of the vibrations of a cantilever under axial load. *Composite Structures* 62:423–428
- Lofberg J (2004) YALMIP: A toolbox for modeling and optimization in MATLAB. In: Proceedings of the CACSD Conference, Taipei, Taiwan
- Lorini G, Miotti A, Scattolini R (2006) Modeling, simulation and predictive control of a spark ignition engine. In: Predimot (ed) Predictive control of combustion engines, TRAUNER Druck GmbH & CoKG, pp 39–55
- Maciejowski J (2002) Predictive Control with Constraints, 1st edn. Prentice Hall
- Maciejowski JM (2000) Predictive control with constraints. University of Cambridge
- Mao X, Wang D, Xiao W, Liu Z, Wang J, Tang H (2009) Lean limit and emissions improvement for a spark-ignited natural gas engine using a generalized predictive control (GPC)-based air/fuel ratio controller. *Energy & Fuels* (23):6026–6032
- MathWorks T (2007) Matlab signal processing blockset v6.6 (r2007b). Software.
- Mayne DQ, Rawlings JB, Rao CV, Sokaert POM (2000a) Constrained model predictive control: Stability and optimality. *Automatica* 36:789–814
- Mayne DQ, Rawlings JB, Rao CV, Sokaert POM (2000b) Constrained model predictive control: Stability and optimality. *Automatica* (36):789–814
- Murray-Smith R, Johanssen TA (1997) Multiple model approaches to modelling and control. Taylor & Francis
- Muske KR, Jones JCP (2006) A model-based SI engine air fuel ratio controller. American Control Conference, Minneapolis, USA
- Polóni T, Rohal'-Ilkiv B, Johansen TA (2007) Multiple ARX model-based air-fuel ratio predictive control for SI engines. In: IFAC Workshop on advanced fuzzy and neural control, Valenciennes, France, conference paper MO5-3
- Polóni T, Johansen TA, Rohal'-Ilkiv B (2008) Identification and modeling of air-fuel ratio dynamics of a gasoline combustion engine with weighted arx model network. *Transaction of the ASME (Journal of Dynamic Systems, Measurement, and Control)* 130(6), 061009
- Preumont A (2002) *Vibration Control of Active Structures*, 2nd edn. Kluwer Academic Publishers
- Rossiter JA (2003) *Model-based predictive control: a practical approach*, 1st edn. CRC Press LCC
- Song G, Qiao PZ, Bibienda WK, Zhou GP (2002) Active vibration damping of composite beam using smart sensors and actuators. *Journal of Aerospace Engineering* 15(3):97–103
- Sturm JF (1999) Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software - Special issue on Interior Point Methods* 11-12:625–653
- Takács G, Rohal'-Ilkiv B (2009) MPC with guaranteed stability and constraint feasibility on flexible vibrating active structures: a comparative study. In: Proceedings of The eleventh IASTED International Conference on Control and Applications, Cambridge, United Kingdom.
- Wills AG, Bates D, Fleming AJ, Ninness B, Moheimani SOR (2008) Model predictive control applied to constraint handling in active noise and vibration control. *IEEE Transactions on Control Systems Technology* 16(1):3–12
- Zeman J, Rohal'-Ilkiv B (2003) Robust min-max model predictive control of linear systems with constraints. *IEEE International Conference on Industrial Technology*, pp 930 – 935

- Zhai YJ, Yu DL (2009) Neural network model-based automotive engine air/fuel ratio control and robustness evaluation. *Engineering Applications of Artificial Intelligence* (22):171–180
- Zhai YJ, Ding-WenYu, Hong-YuGuo, DLYu (2010) Robust air/fuel ratio control with adaptive DRNN model and AD tuning. *Engineering Applications of Artificial Intelligence* (23):283–289

*Comments – Remarks*



*Comments – Remarks*

*Comments – Remarks*

*Comments – Remarks*

# Chapter 6

## Laboratory Model of Coupled Tanks

Vladimír Žilka and Mikuláš Huba

**Abstract** This chapter gives introduction into work with frequently used hydro-system of coupled tanks. The chapter explains analytical modelling for one and two-level systems of coupled tanks, including the experimental identification of system's parameters and sensor's calibration.

### 6.1 Introduction

To be able to use their knowledge in solving practical tasks, for graduates it is important to develop skills in applying control theory related to linear and non linear systems to real plants control. This may be just partially achieved by computer simulations and the feedback received in such a way is never complete. Therefore, it is important to acquire already during the university studies some practice and experience related competences in designing and testing of controllers on real systems.

Due to the simple manipulation, easily understandable and observable processes and their time constants, one of many systems used at universities frequently as pedagogical instruments in control education the system of coupled tanks has to be mentioned. The theoretical design of controllers included in this publication comprehends two basic configurations: control of one, or of two coupled tanks. In considering coupled tanks, one of possible configurations is the coupling in series with interaction, when the liquid's level in one of the tanks influences the other tank's liquid's level as the liquid may flow

---

Vladimír Žilka

Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, e-mail: [vladimir.zilka@stuba.sk](mailto:vladimir.zilka@stuba.sk)

Mikuláš Huba

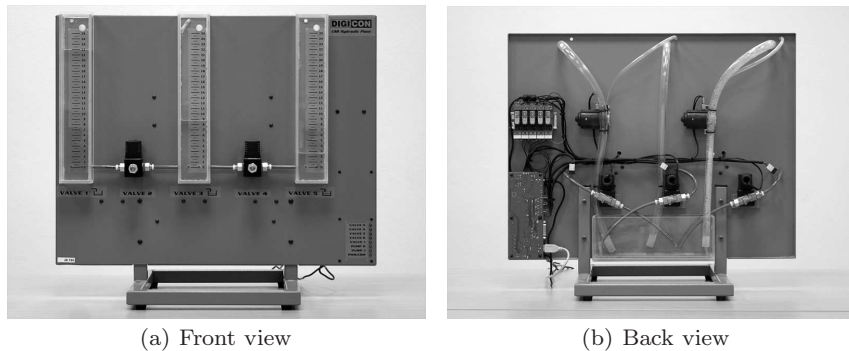
Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, e-mail: [mikulas.huba@stuba.sk](mailto:mikulas.huba@stuba.sk)

between the tanks in both direction. The other two-tank consideration is the cascade coupling with one-way influence, where the tanks are placed one under another. The liquid from the bottom tank can not flow into the upper one, plus the upper tank's inflow impact does not depend on the liquid's level in the lower tank. Due to the available equipment, in the practical part of this chapter we will consider just coupling in series with mutual interaction of tanks.

In order to be able to demonstrate theoretical aims of the control algorithm design and verify real processes by simulation using their models, we need to know the process model and its parameters. Sometimes, the simplest possible model is required, but for some tasks, the most accurate model possible is wellcomed. In case of inaccurate mathematical model, the associated controller (if not sufficiently robust) would function only in simulations. That's why we will be dealing from now with the analytical description of the real scheme necessary for construction of the model, with the identification of the proposed model parameters, as well as with confrontation of the model with the real process and with control processes associated with control design based on considered models.

## 6.2 Coupled Tanks – Hydraulic Plant

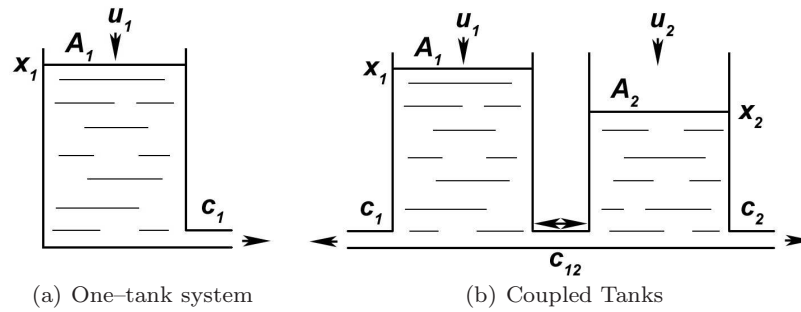
Coupled tanks represent one of the most common examples of non-linear system. The task is usually related to basic manipulation required for dealing with the available physical system. System considered in our publication is shown in Fig. 6.1. This hydraulic system is composed of 2 pumps and 3 coupled tanks, each of them with the own drain. It was designed according to Žilka (2007).



**Fig. 6.1** Coupled tanks uDAQ28/3H2. Front and back view.

The system is easily connectable to the PC via USB interface. There is no need of additional data acquisition cards hence the controlling is performed directly from the Matlab/SIMULINK or Scilab/Scicos environment. By switching the valves between the tanks and between the tanks and the lower reservoir, it enables to create different configurations, e.g. the one-level system, two-levels system with one inflow and one drain, or the multi-input-multi-output (MIMO) system with several inflows and outflows and the mutual tanks' coupling.

At the beginning we will start with the easiest model – we will imagine only one tank where the level of liquid is influenced only by inflow from the pump and drain from the valve. This valve is only two positions (open/close). This kind of connection's model can be seen in Fig. 6.2.



**Fig. 6.2** Models of tanks

The liquid's level is denoted as  $x_1$ , with the liquid's inflow  $u_1$ , tank's cross-sections  $A_1$  and the valve's outflow coefficient  $c_1$ . This tank model can be described by various differential equations acquired through the application of the law of substance preservation. Thus it applies for the given tank that the change in volume  $V$  is defined by the difference between the inflow and outflow. If we know the inflow  $u_1$  and we define the outflow as  $u_{\text{out}}$  then

$$\begin{aligned}\Delta V &= u_1 - u_{\text{out}} \\ V &= x_1 A_1 \\ A_1 \frac{dx_1}{dt} &= u_1 - u_{\text{out}}\end{aligned}$$

To determine  $u_{\text{out}}$  we can use the Toricelli formula which defines the speed of the liquid outflowing the tank's aperture  $v_{\text{out}}$ . With the outflow, we have to take into account the co-actor as well  $\mu$  – for the water 0.63. Plus if we already know the cross-sections the valve's aperture  $A_{\text{vo}}$ , we can put down the following:

$$v_{\text{out}} = \sqrt{2gx_1}$$

$$A_1 \frac{dx_1}{dt} = u_1 - \mu A_{v_0} \sqrt{2gx_1}$$

Now by applying the substitution and defining the flow coefficient of the valve  $c_1$

$$c_1 = \frac{\mu A_{v_0}}{A_1} \sqrt{2g}$$

The resulting differential equation for the one-level system would be

$$\begin{aligned} \dot{x}_1 &= \frac{1}{A_1} u_1 - c_1 \sqrt{x_1} \\ y_1 &= x_1 \end{aligned} \quad (6.1)$$

In this way we can derive the differential equations for coupled tanks. System is then composed of two mutually interacting tanks with two possible pumps. Each tank would have a separate inflow  $u_1$  a  $u_2$ , separate outflow depending on  $x_1$  and  $x_2$  and the flow between the tanks specified by difference of the levels and the constant  $c_{12}$ . Such tank model can be then described by differential equations

$$\begin{aligned} \dot{x}_1 &= \frac{1}{A_1} u_1 - c_{12} \text{sign}(x_1 - x_2) \sqrt{|x_1 - x_2|} - c_1 \sqrt{x_1} \\ \dot{x}_2 &= \frac{1}{A_2} u_2 + c_{12} \text{sign}(x_1 - x_2) \sqrt{|x_1 - x_2|} - c_2 \sqrt{x_2} \\ y_1 &= x_1 \\ y_2 &= x_2 \end{aligned} \quad (6.2)$$

The liquid's levels in the first and second tank are denoted as  $x_1$ ,  $x_2$  and the level surfaces as  $A_1$ ,  $A_2$ . For Simulink model see Fig. 6.3. In block MatlabFcn following equation was made

```
c_{12}sign(u(1)-u(2))sqrt(abs(u(1)-u(2)))
```

### 6.2.1 Identification

For simulating or controlling the real system it is indispensable to know all parameters of its mathematical model. While the measurement of the level surfaces in tanks does not represent a serious problem, to determine the values of the valve's flow coefficient is usually a bit more difficult. By taking the more detailed view on (6.1) or (6.2) we will realize that we can define at least two approaches for determining parameters  $c_1$ ,  $c_2$  and  $c_{12}$ . One is

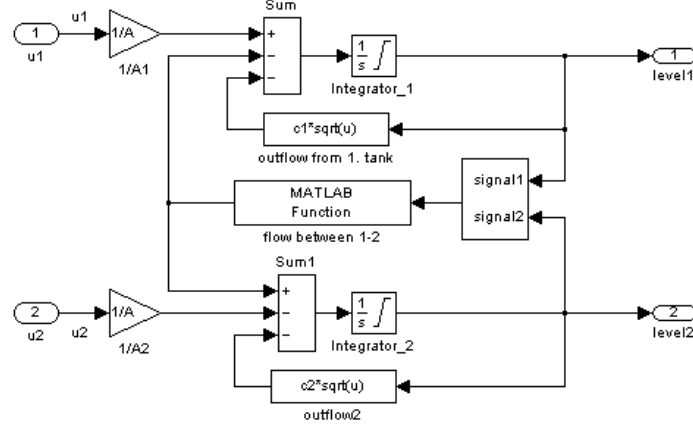


Fig. 6.3 Coupled tanks - Simulink model.

based on measurement with constant inflow, the other on experiment with zero inflow – tank’s emptying.

In the first approach based on measuring steady state under constant inflow the resulting state equation reduces to (6.1) is

$$\begin{aligned} \frac{1}{A_1}u_1 &= c_1\sqrt{x_1} \\ c_1 &= \frac{u_1}{A_1\sqrt{x_1}} \end{aligned} \quad (6.3)$$

The second approach is based on evaluating transients under zero inflow – i.e. during tanks’ emptying. The flow coefficient  $c_1$  of the valve can be easily obtained from (6.1). If the pump is off, the state equation (6.1) reduces to

$$\begin{aligned} \dot{x}_1 &= -c_1\sqrt{x_1} \\ y_1 &= x_1 \end{aligned}$$

In this case, by integrating above equation,  $c_1$  can be get as

$$c_1 = \frac{2\sqrt{x_{1\text{init}}} - 2\sqrt{x_{1\text{final}}}}{\Delta t_1} \quad (6.4)$$

where  $x_{1\text{init}}$  represents the initial level of the liquid before opening the valve and  $x_{1\text{final}}$  its final level when we finish experiment by closing the valve. Finally,  $\Delta t_1$  represents the duration of the experiment.

The second way of determination  $c_1$  does not require to know the exact inflow from the pump, which can leads to higher accuracy. This way is also



faster. Time required to prepare and perform the measurement takes just about minute.

In both cases it is convenient to perform several measurements at different level heights  $x_1$  and to define the resulting value  $c_1$  as a mean value. It is caused mainly by the fact that the solenoid valves do not have aperture of a simple circular shape, which, together with the relatively low pressure and low level height may lead uncertain and to slightly fluctuating values of the valve constant  $c_1$ .

Indetermining the valve constant  $c_{12}$  describing interconnection between the tanks we can use similar approach. The difference would only be in the fact that during the experiment the outflow valves from tank 1 and 2 will be closed and before its beginning one tank will be filled to the maximum level and the other one made empty. We stop the measurement just before the complete alignment of both levels. The resulting relation defined from (6.2) for  $c_{12}$  will be

$$c_{12} = \frac{\sqrt{|h_{10} - h_{20}|} - \sqrt{|h_{11} - h_{21}|}}{\Delta t_{12}}$$

With  $h_{10}, h_{20}$  as the initial liquid's levels in the first or second tank,  $h_{11}, h_{21}$  as the final liquid's levels after finishing the experiment and  $t_{12}$  as the duration of the measured interval.

It is important to mention that in case with the tank's outflow orifice not being exactly at the same altitude as the valve's drain orifice (but lower), we have to consider this difference. Then it is to consider that the real liquid's level in the tank producing the resulting pressure at the valve orifice is

$$x_1 = x_{1\text{tank}} + x_{1\text{offset}}$$

where  $x_{1\text{tank}}$  represents the height of the water column in the first tank and  $x_{1\text{offset}}$  denotes the offset between tank orifice and valve outflow.

The pump identification seems to be easier. It may be based on determining time periods, in which the tank with closed valves will be completely filled under consideration of different pump's input voltage.

$$q_1 = \frac{A_1 x_1}{\Delta t_1}$$

It is vital to choose various working points in whole pump's working scale. Finally, since the pump is a reasonably nonlinear element, we approximate the measured pump characteristic by the curve fitting methods, e.g. with the polynomials of third or fourth degree. The input-output characteristic of the first pump, the inverse input-output characteristic of the pump and the characteristic of the first valve are depicted in Fig. 6.6.

### 6.2.2 Sensors Calibration

To measure the height of liquid's level in the plant in Fig. 6.1 the pressure sensors are used. These have several advantages: they are easy to change in a case of failure, they don't require any maintenance, they don't corrode and water sediments don't attach to them. On the other hand, the measurement depends on the surrounding pressure, thus it is recommended to perform at least at the beginning of the measurement day, the calibration of these sensors. It consists of the definition of offset and sensor's gain values, what should be done for each liquid's level separately. The dependency is linear. The basic scheme of the conversion of raw signals on liquid's levels in meters can be found in Fig. 6.4. (All the necessary parameters for signal adjustment lay in one array denoted as *calibParams*. Array "pumps" contains the pump's entrances (performance in % from 0% to 100%) and field "valves" contains the valves configuration 0 – close, 1 – open, for each valve.)

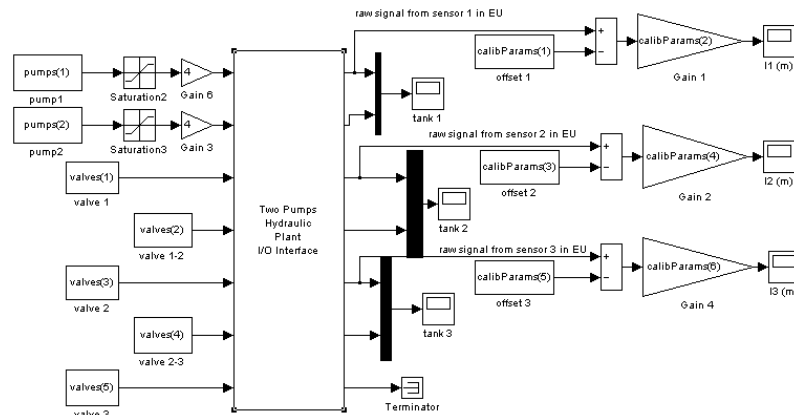


Fig. 6.4 Basic diagram for calibration and configuration.

Under manual calibration we will partially fill the calibrated tank and empty it subsequently. Raw integer number from each output is sensor offset  $x_{1\text{empty}}$ . This is specified in engineering units (EU). Afterwards, we fill up the tank to the maximum height of liquid's level and we define the output's value  $x_{1\text{empty}}$  in EU. We deduct, once again, the value of sensor's outputs. The conversion to liquid's level in meters is defined by the relation

$$\begin{aligned} \text{sensorGain}_1 &= \frac{(x_{1\text{full}} - x_{1\text{empty}})}{\Delta x_1} \\ x_1 &= \frac{x_{1\text{actual}} - x_{1\text{empty}}}{\text{sensorGain}_1} \end{aligned} \quad (6.5)$$

Here,  $x_{1\text{actual}}$  is the actual value of liquid's level in EU,  $x_{1\text{empty}}$  is the output in EU at the zero liquid level and  $x_{1\text{full}}$  is the value of the output in EU at maximum liquid level and  $\Delta x_1$  is the height of the maximum liquid level. For example, if we know that at the zero level  $x_{1\text{empty}} = 800$  EU, at the maximum level  $\Delta x_1 = 0.25$  m and output 3300 EU, for the output 2300 EU it holds:

$$\begin{aligned} \text{sensorGain}_1 &= \frac{3300 - 800}{0.25} = 10000 \text{ EU m}^{-1} \\ x_1 &= \frac{2300 - 800}{10000} = 0.15 \text{ m} \end{aligned}$$

After the calibration it is suitable to perform identification of system's parameters. It consists of the estimation of the flow coefficients of valves and of the pump's parameters according to the above-mentioned process.

### 6.2.3 Automatic Calibration and Identification

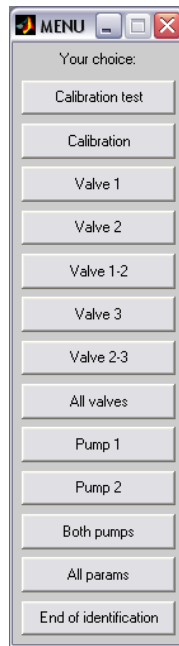
The above mentioned procedure of calibration and identification is obviously time consuming and also not easy to calculate manually. To perform the such long time step-by-step measurements of the pump's characteristics plus to save the measured values subsequently after each measurement and evaluate them afterwards, we have to sacrifice an enormous quantity of time. That's all is the reason why we developed the software package for automatic identification of system's individual parameters and for identification of all valve and pump parameters. The control is very easy – only one click is needed to get the required identification in the menu. Menu's environment can be seen in Fig. 6.5.

As the processes were described in details in the previous chapter, the individual functionalities will be mentioned now just shortly. At the identification of the valve 1,2 or 3, the tank is fill up to the height of approximately 24.5 cm, then we wait till the liquid's level is stabilized and the tank is afterwards emptied to 0.8 cm. (At almost zero level, the valve's behaviour is strictly non-linear and as we are not planning to regulate the level under 1 cm, we don't consider the lowest boundary). The measured values are saved, together with all used constants, into the mat file in the following format:

`valveValvenumber-year-month-day-hour-minute-second.mat.`

For example: `valve1-2009-12-20-16-26-43.mat` and so on. We don't consider the first and the last measurement sample, from the other values with the step approximately 1.5 cm – we use each sixth sample, define the value  $c_1$  as well as the possible offset value of the valve  $offset_1$ .

At the end we define the mean value of offset and of flow coefficient and we save those two values together with the time course of emptying into the `valve1`, `valve2` and `valve3.mat`. We apply them directly to the proposed model, we simulate the emptying and we draw the comparison of simulation

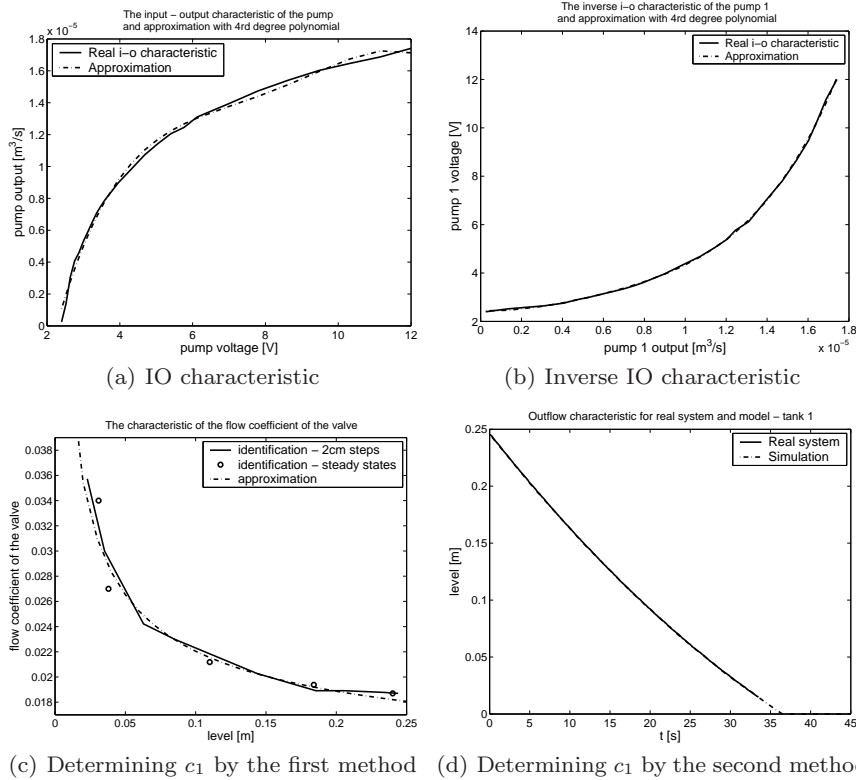


**Fig. 6.5** Menu of the software module for calibration and identification.

and real data into the graph of emptying characteristics such as Fig. 6.6. This procedure allows us to check visually quality of the valve's identification on the graph. The registration to the actual mat file allows us to read variables in the program according to our choice, at the direction of the model or at the work with the real system.

The previous identification may sometimes look better than the new one – that's why the measurements are stored in the mentioned file with date and time – that way we are able to see the history at any time. We apply the similar procedure at the other measurements as well.

Basically in the same way we identify the valve between the tanks. After the measurement is launched, the first tank is filled and the second emptied, the interconnecting valve is opened afterwards and the measurement is performed until the leveling of two liquid's levels. Then we perform the second measurement, the second tank is filled up and the first emptied. We identify again the flow coefficient and we use those two measurements to define the mean coefficient  $c_{12}$ . We save the measured values, following the above-mentioned procedure, just this time the mat file will be named valve12.mat – for the actual value and valve12day-time.mat for archiving. We'll use the identified coefficient for the simulation and we draw the comparison of model's course and the real into the graph. In case of lesser compliance, we repeat the procedure.



**Fig. 6.6** Measured Input-Output pump characteristic and its approximation by the 4th degree polynomial and inverse input-output pump characteristic and its approximation by the 4th degree polynomial (above). Determination of the valve coefficient  $c_1$  with two approaches – comparing the measurement and approximative data for tank 1 (below).

The measurement of pump's characteristics is more time-consuming. In order to achieve the required accuracy, we continue with increasing the pump input by the step 3% of the maximal range 100% up to the moment where the pump starts slowly to draw the liquid. (In our case it will be around 19 – 21 %.) At lower inputs, the pump is not working.

At the beginning of measurement, we empty the tank, and then fill it up by given input up to the fullest state, or during 60 seconds. That's because at the lowest levels of input, the filling up would take too much time. After the last measurement – for full input, we define the maximum inflow of the pump as well. We put the given inflows and power tensions into the field and save them, at the end of experiment, by same way as valves, into the mat files. The first pump's name will obviously start by pump1 and the second one by pump2.

During the individual steps, we include into the measurement the artificial break of 40 s, to prevent the changes in pump's parameters, its heating or to partially isolate the previous measurement from the actual one. We draw the measured IO or inverse IO characteristics into the graph and compare them with the approximations acquired thanks to curve fitting functions.

As the buttons' names indicate, by the above-described procedures we can identify all valves, both pumps and all system's parameters.

To clean the older data, in order to delete the measured and saved values more easily, we can use the script *cleaning* which will preserve only the measured values and will delete all the mat files that contain the date in the name. That's why one has to be very careful when using it. We should use it only if we are sure, that we don't want to archive the older values.

After we get all the required parameters, we only have to load them and have a model that will use them correctly. The basic program would afterwards look as follows:

```
load 'calibParams'; % offsets and gains of pressure sensors
load 'valve1';      % parameters of drain valve from tank 1
load 'valve2';      % parameters of drain valve from tank 1
load 'valve12';     % parameters of valve between tank 1 & 2
load 'pump1';       % pump 1 parameters
load 'pump2';       % pump 2 parameters
A = 0.001;          % tank area
T = 0.25;           % sampling period
open('model.mdl'); % open simulink model
```

Pump's subblock is depicted in Fig. 6.7 where function block 'U->Q1' contains:

$$PU1(1)*u(1)^4+PU1(2)*u(1)^3+PU1(3)*u(1)^2+PU1(4)*u(1)+PU1(5)$$

This 4 degree polynomial enables linearization of the pump. The input for this function is inflow from controller (in  $\text{m}^3 \text{s}^{-1}$ ). Function convert this input to Volts. The 'Gain4' changes Volts to %, because the pump input in matlab block is in %.

There is also treshold. For too small input (smaller than  $10^{-6} \text{m}^3 \text{s}^{-1}$ ) push to output 0. This feaure respects the fact that the pump can not process smaller inputs.

### 6.2.4 Some Recommendation for Users

At the end we would like to add some recommendations that may help you in keeping your plant in optimal work.

Use always distilled water, even in the case you need to add just small amount of it. In this way you may reasonably prolonged intervals between clearings.

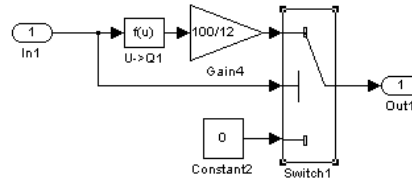


Fig. 6.7 Pump implementation in simulink.

For removing sediments from the walls of the tanks use a soft brash appropriate for bottle cleaning. Pump out the water containing such sediments and flush the system several times by clean water. After that you need to pump out also the standard water used for cleaning and again fill the container by the distilled one.

For eliminating the evaporation fill in the water into tanks that are covered and have smaller evaporation area.

After a longer period without use it may happen that some valve does not open or close. In such a case it is necessary to repeat open/close procedure several times (even when it does not function) and then slightly hit the metal screws in the valve centre. This may help to release the valve.

When you know that the system will not be used for a longer time, pump out all water.

When you keep all these rules, the system does not require any maintenance. Only in extreme situations when the system was not used for longer time period without pumping out all water, the sediments may not only cover the walls of containers, but even tubes, pressure sensors and valves. When facing problems with pressure sensor, after emptying the containers by a gentle pull remove the sensor from the tube and clean the tube by a thin wooden tool. When the sediments clog up the valve, it is necessary to unscrew fittings from both sides and to flush the valve under pressure, e.g. by a rubber ball filled by water. As it was, however, mentioned above, in case of regular maintenance and storing such problems will not occur.

**Acknowledgements** The work has been partially supported by the Slovak Grant Agency grants No. VG-1/0656/09 and VG-1/0369/10. It was also supported by a grant (No. NIL-I-007-d) from Iceland, Liechtenstein and Norway through the EEA Financial Mechanism and the Norwegian Financial Mechanism. This project is also co-financed from the state budget of the Slovak Republic.

## References

- Žilka V (2007) Nonlinear controllers for a fluid tank system. Master's thesis, Faculty of Electrical Engineering and IT, Slovak University of Technology, Bratislava, Slovakia, (in Slovak)



*Comments – Remarks*

*Comments – Remarks*

*Comments – Remarks*

*Comments – Remarks*



# Chapter 7

## Constrained PID Control Tasks for Coupled Tanks Control

Vladimír Žilka and Mikuláš Huba

**Abstract** This chapter treats basic issues of linear and non-linear controller's design applied to control of hydraulic plant of available three-tank-system. Controllers based on approximation of the given non-linear system by the first order linear system by step responses are proposed and compared with controllers based on the analytical linearization of the nonlinear model in a fixed operating point, by the input-output linearization in the actual state (exact linearization) and around a generalized operating point are developed and verified for the one-level and two-level system.

### 7.1 Introduction

A plant to be controlled may be modelled and identified in many ways. It is e.g. possible to derive mathematical models of the plant that for a given data show nice matching but when used for the controller design, the resulting performance will be pure. In case of inaccurate, or inappropriate mathematical model, the resulting regulators would function only in simulations. That's why we will be dealing from now with the description of the real plant, estimation of its model, identification of its parameters as well as with confrontation of results achieved in control of the plant with the performance required in the controller design.

In the next paragraph, we will show some examples of simple P, PI and PD controllers for different tanks' configurations, from the simplest one tank

---

Vladimír Žilka

Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, e-mail: [vladimir.zilka@stuba.sk](mailto:vladimir.zilka@stuba.sk)

Mikuláš Huba

Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, e-mail: [mikulas.huba@stuba.sk](mailto:mikulas.huba@stuba.sk)

system, with closed output valves, through one-level systems with one inflow and one outflow up to two-level system linearized by generalized input-output linearization and controlled by the PD<sub>2</sub> controller.

## 7.2 Basic P and PI controllers

At the beginning, it would be the best to introduce the simplest case of hydraulic system's configuration. In doing so we will consider one-level system which has all the valves closed. The pump feeding the liquid into this tank will represent the actuator. Under control, such system behaves as simple integrator. Considering 6.1 we will define differential equations of such system in the following form:

$$\begin{aligned} \dot{x}_1 &= \frac{1}{A_1} u_1 \\ y_1 &= x_1 \end{aligned} \quad (7.1)$$

Obviously, we are dealing with linear system with the transfer function

$$F_1(s) = \frac{Y_1(s)}{U_1(s)} = \frac{1}{A_1 s}$$

i.e. with integrator having the integral time constant  $A_1$  (gain  $1/A_1$ ). From theory we know that to stabilize such a system (7.1), P-controller with a gain  $P_1$  would be sufficient. The closed loop transfer function will be:

$$G(s) = \frac{P_1 F_1(s)}{1 + P_1 F_1(s)} = \frac{P_1}{A_1 s + P_1}$$

If we want to control this system, we can request it to behave as the first order linear system with the transfer function

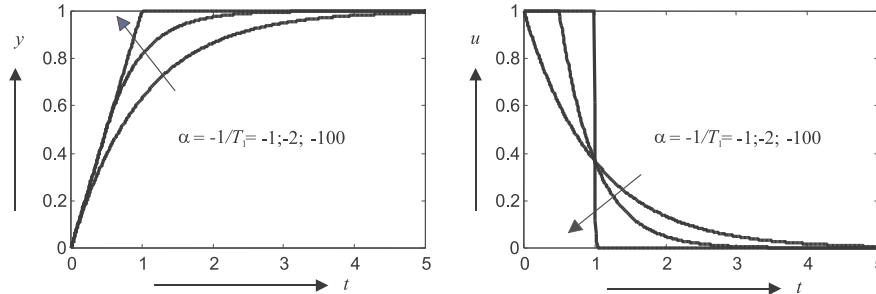
$$G(s) = \frac{1}{T_1 s + 1}$$

In such a case, the P-controller's gain results as

$$P_1 = \frac{A_1}{T_1}$$

It is clear that by choosing the time constant  $T_1$ , we are able to define the dynamics of the closed loop that might also be expressed by its closed loop pole  $\alpha = -1/T_1$ . By decreasing this time constant, when the closed loop pole is being shifted to  $-\infty$ , we can speed up the transients processes. And contrary, by increasing  $T_1$ , when the pole is shifted towards the origin, we

can slow down them. In increasing the P-controller gain we have to be careful and speed-up the transients only until certain limit.



**Fig. 7.1** Impact of the P controller tuning on the transient responses

The limitations are mainly given by factors as nonmodelled dynamics, plant uncertainty, measurement noise and control signal saturation. For shorter time constant  $T_1$ , saturation of the control signal typically occur and for  $T_1 \rightarrow 0$  the control signal tends to rectangular pulse of the minimum time control (Fig. 7.1). P-controller for integrator plant represents solution of the dynamical class 0.

If we increase the amplification of controller too much, the sensitivity of the circuit to the measurement and quantisation noise will increase and this can lead to system's destabilization, increased wear of active elements or damage – in this case, by switching the pump on and off over a too long period to maximum or zero voltage.

On the other hand, too small controller gain can slow down the transient processes, or, in an extreme case, the pump will even not start to move. In order to get some insight, which controller gain may be applied without leading to parasitic oscillations, it is necessary to approximate the non-modelled loop dynamics that can be characterized by equivalent dead time  $T_d$ . This can be identified as the time between a control signal step and beginning of the reaction of the measured output. Then, it is recommended not to use shorter time constants (“faster” poles) as those given by

$$T_1 = e^1 T_d; \alpha = -1 / (e^1 T_d) \quad (7.2)$$

For the given hydraulic system, notice that the pump is reacting only to positive values of the input voltage. So, its control is only possible in one way – to increase the liquid's level in tank. That's why it is necessary to empty the tank before the experiment, or after it.



### 7.2.1 PI-controller

Now, let's consider situation with opened outflow valve characterized with the flow coefficient  $c_1$ . In this case the one-level system will be in the form 6.1. This system does no longer behave as pure integrator. Let's try to control it, at the beginning, with P-controller, designed in the previous part. By examining few operating point, we will realize that P-controller can't control this system without the permanent control error. It is no surprise, if we take into account that the opened outflow requires permanent inflow, but the zero control error leads to zero inflow. With zero control error generates the P-controller zero action, but considering the fact that the liquid outflows from the tank, it decreases the liquid's level below the desired values. To compensate the outgoing liquid we have to use more advanced controllers.

The easiest way to compensate the outgoing water is to add parallel signal to the output of the controller. Such constant signal set usually manually was originally called as reset (offset). Later, this was replaced by *automatic reset* denoted today as the *integral action* and produced directly by the controller. By this modification we will obtain the PI-controller. The goal of the integral action is to integrate the control error values multiplied by a suitable constant what will continuously increase the controller output – until the complete removal of the control error. The transfer function of such PI-controller is:

$$R(s) = P + \frac{I}{s}$$

where  $P$  represents the gain of the proportional action of the controller and  $I$  is the gain of the integral action.

An easy experimental way to set up the PI-controller's parameters may be based on measured step-responses of the system. The step-responses of the system contain information about the basic dynamic attributes of the system and this is why it enables us to set up the controller's parameters better than by the frequently used *trial and error* method. At the same time, we have to realize that the step-response method was developed for the responses of linear system to a unit step. Our system (6.1) is, however, a nonlinear one. We can't thus talk about the system's step-responses for the whole scope of liquid's levels. The obtained information will depend on the choice of concrete operating point, in which the step will be measured. For the PI-controller design having the best attributes in the most ample areas, it may seem to be useful to choose the operating point approximately in the middle of the working area.

In the surroundings of chosen operating point we will approximate our non-linear system (6.1) by the first order linear model. This will be obtained by the step-response method from step responses measured firstly by setting up by appropriate input pump voltage the liquid's level approximately to the middle of the tank height and here to stabilize it for some time, optimally few centimeters under the middle. When the liquid's level will be stabilized,

we will switch the pump input to higher voltage by approximately 1V step and wait again for the liquid's level to stabilize (it should stabilize few centimeters over the middle). From the measured step-response we will find out by constructing tangent at the origin the approximate value of time constant  $T_{1n}$  and the value of the approximative plant gain  $K_1 = \Delta y / \Delta u$ .

So, in a close surroundings of the operating point corresponding to the input voltage about  $u_1 = 4V$  the system may be approximated by linear transfer function

$$F_1(s) = \frac{K_1}{T_{1n}s + 1}$$

Then, in the previous paragraph mentioned P-controller design may now be used as well. When the open loop transfer function was in the case of the P-controller design given as  $1/T_1s$  the closed loop had the transfer  $1/(T_1s + 1)$ . Now, we will design controller to yield the same open loop transfer function as previously, i.e.  $1/T_1s$ , what yields

$$R_1(s) = \frac{1}{T_1s} \cdot \frac{1}{F_1(s)} = \frac{T_{1n}s + 1}{K_1T_1s} = \frac{T_{1n}}{K_1T_1} + \frac{1}{K_1T_1s}$$

Thus the  $P$  and  $I$  actions will be

$$P_1 = \frac{T_{1n}}{K_1T_1}$$

$$I_1 = \frac{1}{K_1T_1}$$

After calculating and testing this tuning we recommend you to experiment a bit and to track the influence of the changes of  $P_1$  and  $I_1$  on the control performance. You may observe that the achieved dynamics is different from the one possible pulse at saturation that was typical for the P controller design. This solution represents basic part of the PI controller of the dynamical class 0 (it should yet be completed by a prefilter with the time constant  $T_{1n}$  – then the control signal after a setpoint step exponentially increases monotonically to its new steady-state value). Attempts to achieve faster dynamics may be connected with strong overshooting of the output signal.

### 7.2.2 $PI_1$ controller

Improved dynamics for larger setpoint steps without producing windup effect is possible to achieve by the so called  $PI_1$  controller that typically has one interval of the control signal at the saturation limit. This controller is based on reconstruction of the equivalent input or output disturbances by appropriate observer using inverse or parallel plant model. Since the hydraulic plant (6.1)

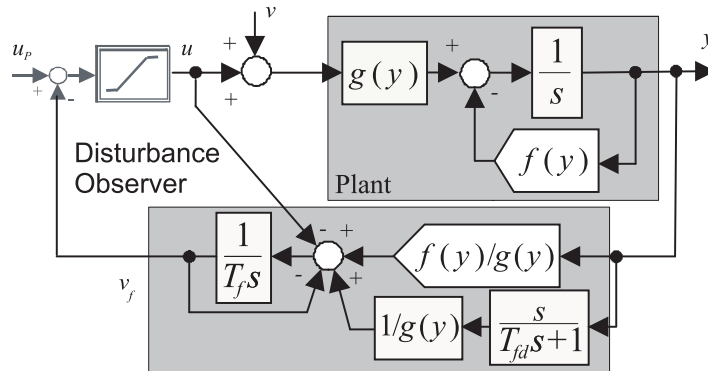
is stable, inversion of the plant dynamics may be directly derived also for the nonlinear model as

$$\hat{u}_1 = A_1 [\dot{y}_1 + c_1 \sqrt{y_1}] \quad (7.3)$$

The input disturbance  $v$  may then be reconstructed by comparing the reconstructed plant input  $\hat{u}_1$  and the controller output according to

$$\hat{v} = \frac{\hat{u}_1 - u}{1 + T_f s} \quad (7.4)$$

and then compensated at the P-controller output  $u_P$ . Such a compensation may be treated as a special case of nonlinear disturbance observer in Fig. 7.2. Besides of the P-controller gain, the new tuning parameter  $T_f$  may be used to increase system robustness against uncertainties and non-modelled dynamics and to filter measurement noise.



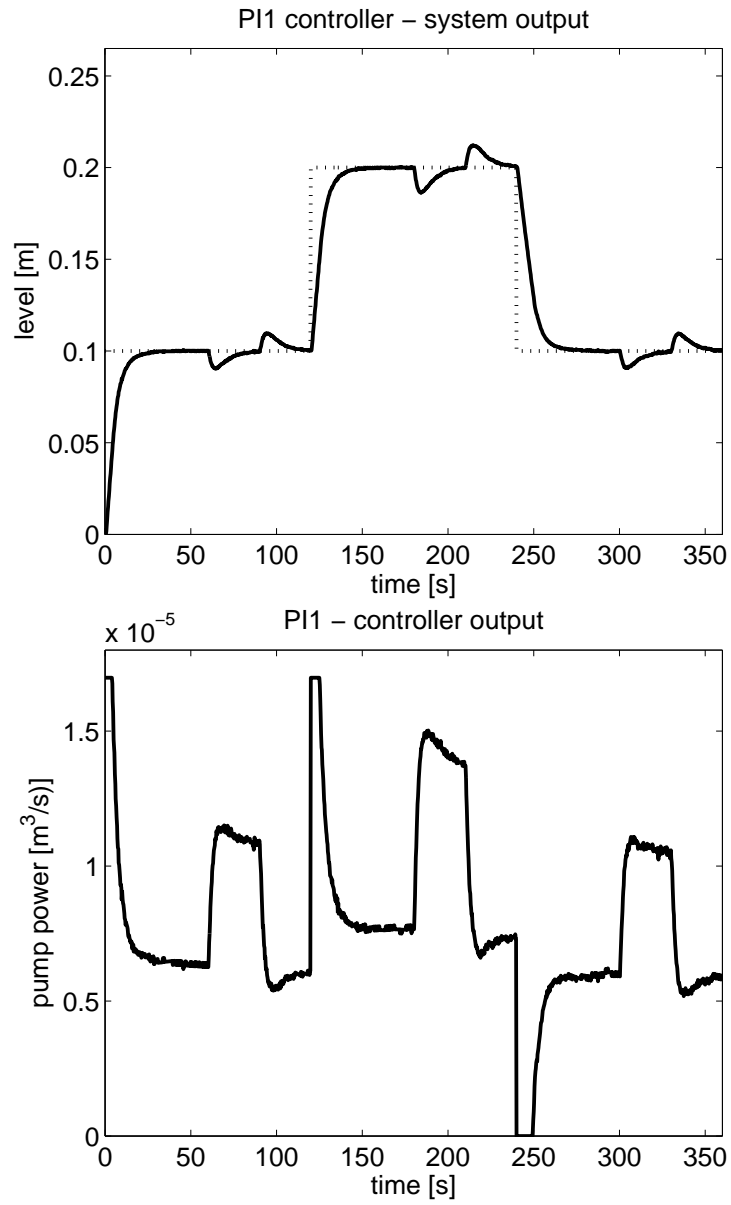
**Fig. 7.2** Rejection of the input disturbances  $v$  by correction of the P controller output in the nonlinear  $PI_1$  controller for system  $dy/dt = g(y)(u + v) - f(y)$

You may test your integral controller by filling to different levels also the second tank and then opening or closing the interconnecting valve among the tanks.

The controller structure in Simulink is depicted in Fig. 7.4. Control program, which work with results of automatic calibration and identification is listed below.

### 7.2.2.1 Programs for Matlab/Simulink

```
clc;
clear all;
close all;
```



**Fig. 7.3** Typical setpoint and disturbance steps in the loop with  $\text{PI}_1$  controller contain one step at saturation after setpoint steps

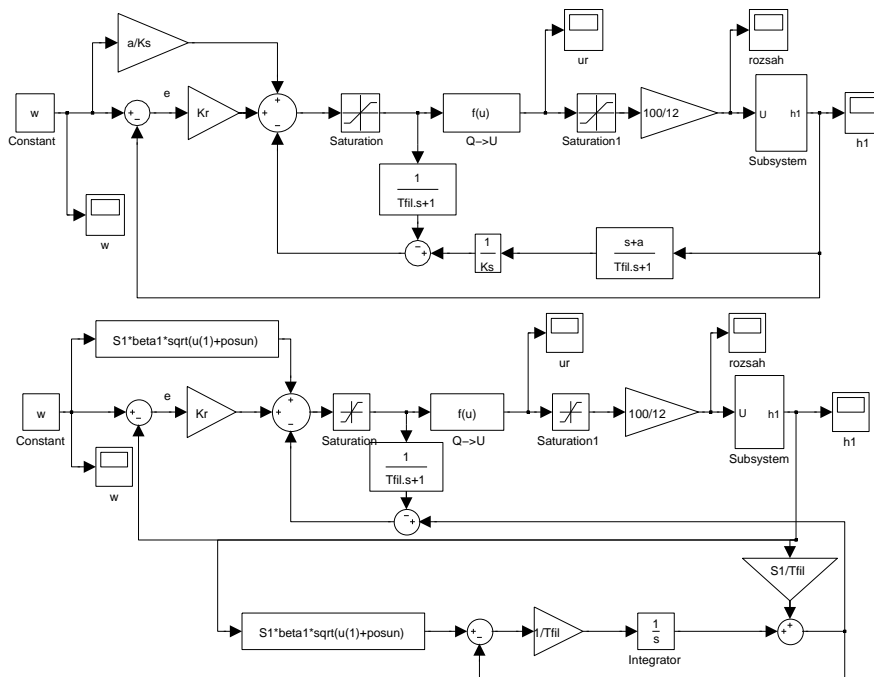


Fig. 7.4 Control structure for linear and nonlinear PI1 implemented in Matlab / Simulink

```

load '../basics/calibParams'; % sensor calibration
                             parameters
load '../basics/valve1';    % parameters of the valve 1
load '../basics/valve2';
load '../basics/valve12';
load '../basics/pump1';    % parameters of the pump 1

T = 0.25;                    % sampling period

S1=0.001;                   % tank area
posun = posun1;             % offset 1

beta1=c1;                   % output valve coeff.
beta12 = 0;
beta2 = 0;

Qmin=0;
Qmax=q_max1;               % control signal constraints

```

```

w11=0.10;          % setpoint values for tank 1 level - 3 steps
w12=0.20;
w13=0.10;

Td = 0.996;       % dead time

Ks = 1/S1;        % plant gain

h10 = w11;        % operating point choise

a = beta1/(2*sqrt(h10+posun)) % linearization around operating
    point

alfae=-0.2;       % directly given closed loop pole
% alfae = -(exp(-(1+a*Td))/Td+a)
                % closed loop pole based on dead time
                estimate
% alfae = -(1+a*Td*exp(1+a*Td))/(Td*exp(1+a*Td))

Kr = -(a+alfae)/Ks; % P controller gain
Tfil = -1/alfae;   % obsever filter time constant

c1 = beta1/(2*sqrt(h10+posun))

Gp = tf([Ks],[1 c1], 'inputdelay',Td)
    % plant model around the operating point c1=a

sim('PI1real2_zilka.mdl');
str_datum = num2str(floor(clock));
str_datum = (strrep(str_datum, '____', '-'));
file_name = sprintf('reg_PI1-%s', str_datum);
eval(['save_', file_name]);

```

### 7.3 Linearization around a fixed operating point

Analytical linearization around a fixed operating point shows one of the simplest ways of designing controller for the one level system. Linearized model may be determined directly from the system's equation (6.1). For a chosen working point and its closest environment, the non-linear system will behave as linear and therefore we can use this as the basic possibility for deriving linear controllers, e.g. the PI one.

While making this linearization we talk about approximation of the non-linear system

$$\begin{aligned}\dot{x}_1 &= f(x_1, u_1) \\ y_1 &= g(x_1)\end{aligned}$$

by a linear system

$$\begin{aligned}\Delta x_1 &= \mathbf{A}\Delta x_1 + \mathbf{B}\Delta u_1 \\ \Delta y &= \mathbf{C}\Delta x_1\end{aligned}$$

where  $\Delta x_1 = x_1 - x_{1_0}$ ,  $\Delta u_1 = u_1 - u_{1_0}$ ,  $\Delta y_1 = y_1 - y_{1_0}$  and  $x_{1_0}$ ,  $u_{1_0}$ ,  $y_{1_0}$  is our fixed operating point. It is to stress that the linearization is considered around a steady state, state, input and output are replaced by deviations from it as:  $\Delta x_1$ ,  $\Delta u_1$  and  $\Delta y_1$  and the matrices A, B and C are in general given as:

$$\mathbf{A} = \left( \frac{\partial f}{\partial x_1} \right)_{x_1=x_{1_0}, u_1=u_{1_0}}, \mathbf{B} = \left( \frac{\partial f}{\partial u_1} \right)_{x_1=x_{1_0}, u_1=u_{1_0}}, \mathbf{C} = \left( \frac{\partial g}{\partial x_1} \right)_{x_1=x_{1_0}, u_1=u_{1_0}}$$

In case on one level hydraulic system (6.1) we get by such a linearization

$$\begin{aligned}\Delta \dot{x}_1 &= \frac{1}{A_1} \Delta u_1 - \frac{c_1}{2\sqrt{x_{1_0}}} \Delta x_1 \\ \Delta y_1 &= \Delta x_1\end{aligned}\tag{7.5}$$

We would like to point that (7.5) is not valid for  $x_{1_0} = 0$ . Taking into account that our system (7.5) is linear, we can consider its velocity transfer function

$$F_1(s) = \frac{\Delta Y_1(s)}{\Delta U_1(s)} = \frac{\frac{1}{A_1}}{s + \frac{c_1}{2\sqrt{x_{1_0}}}}$$

Now for this system we propose PI-controller (for the first order linear system). Transfer function of the controller should again be made in such a way to yield an open transfer function  $1/T_1 s$  what may e.g. be achieved by choosing

$$R_1(s) = \frac{1}{T_1 s} \cdot \frac{1}{F_1(s)} = \frac{s + \frac{c_1}{2\sqrt{x_{1_0}}}}{\frac{1}{A_1} T_1 s} = \frac{A_1}{T_1} + \frac{A_1 c_1}{2T_1 \sqrt{x_{1_0}} s}$$

where

$$\begin{aligned}P_1 &= \frac{A_1}{T_1} \\ I_1 &= \frac{A_1 c_1}{2T_1 \sqrt{x_{1_0}}}\end{aligned}$$

are gains of  $P_1$  a  $I_1$  action of the controller. Note that the time constant  $T_1$  can be easily obtained from step-response of the system. We should not

omit the fact, that by designing the controller, we proceed from the system (7.5), which works with deviations from steady state, not with the absolute variables. That's why the controller is using them as well, i.e. it holds

$$R_1(s) = \frac{\Delta U_1(s)}{\Delta E_1(s)}$$

where  $\Delta U_1(s)$  and  $\Delta E_1(s)$  are Laplace transforms of the deviations of the controller output,  $\Delta u_1 = u_1 - u_{1_0}$  and of the control error,  $\Delta e_1 = e_1 - e_{1_0}$ , from steady states  $u_{1_0}$  and  $e_{1_0}$ . Of course, for control error in steady state is  $e_{1_0} = 0$ . In this case  $\Delta e_1 = e_1$ , but at the controller output one still has to consider deviation signal  $\Delta u_1$ . The system (6.1), which we are in fact controlling, has as its input directly  $u$ . So, we need to arrange the controller output by adding the stabilized steady-state value  $u_{1_0}$  and thus obtaining directly  $u_1$

$$u_1 = \Delta u_1 + u_{1_0}$$

The value  $u_{1_0}$  represents the stabilized value of the pump's voltage, at which the liquid's level or liquid's volume in the tank has the stabilized value  $x_{1_0}$ . The numerical value  $u_{1_0}$  is, of course, needed at the implementation of the controller. We'll define it based on system's (6.1). In a steady-state all time derivatives are zero, thus  $\dot{x}_1 = 0$ . After substituting into (6.1) we'll obtain  $u_{1_0}$  defined as

$$u_{1_0} = A_1 c_1 \sqrt{x_{1_0}}$$

## 7.4 Exact Feedback Linearization

In this task, we would like to control the liquid's level for different operating points. By appropriate gain scheduling it is possible to have various operating points and the associated controllers and then to switch between them, but as more practical it seems to schedule the controller parameters continuously. This is usually done by the input-output feedback linearization. In general, it is supposed a non-linear system in the following form:

$$\begin{aligned}\dot{x} &= f(x, u) \\ y &= g(x)\end{aligned}$$

where the state  $x \in \mathbf{R}^n$ , the input  $u \in \mathbf{R}^m$ , the output  $y \in \mathbf{R}^p$ . Find, if possible, a static state feedback

$$u = \varphi(v, x)$$



in such a way that the system would be from point of view of new input  $v$  and output  $y$  linear. For the solution to exist, it has to be given that the system's relative degree  $r_i$  is natural number. The system's relative degree  $r_i$  is a natural number that defines the number of output's derivations  $y_i$  we have to perform in order to obtain the direct dependence on the output  $u$ . In case of non-existence of such number, we say that the output has the infinite relative degree.

$$r_i = \min\{k \in N; \frac{\partial y_i^{(k)}}{\partial u} \neq 0\}$$

More information can be found for example in [Conte et al \(2007\)](#).

For system (6.1) we may say

$$\dot{y}_1 = \frac{1}{A_1}u_1 - c_1\sqrt{x_1} \quad (7.6)$$

that it has the relative degree 1. But also the system (6.2) has the relative degree 1 because

$$\begin{aligned} \dot{y}_1 &= \frac{1}{A_1}u_1 - c_{12} \operatorname{sign}(x_1 - x_2)\sqrt{|x_1 - x_2|} - c_1\sqrt{x_1} \\ \dot{y}_2 &= \frac{1}{A_2}u_2 + c_{12} \operatorname{sign}(x_1 - x_2)\sqrt{|x_1 - x_2|} - c_2\sqrt{x_2} \end{aligned}$$

the basic condition is satisfied. The problem of the static-state feedback linearization is solvable if

$$\operatorname{rank} \frac{\partial(y_1^{(r_1)}, \dots, y_p^{(r_p)})}{\partial u} = p$$

where  $r_i$  are the relative degrees of outputs  $y_i$ , for  $i = 1, \dots, p$ . Static-state feedbacks can be found by solving equation

$$y_i^{(r_i)} = v_i$$

The proof, technical details and additional references can be found in [Conte et al \(2007\)](#).

For the system (6.1), the static-state feedback that solves the linearization problem can be found by solving the equation

$$\dot{y}_1 = v_1$$

for  $u_1$  that is

$$u_1 = A_1 v_1 + A_1 c_1 \sqrt{x_1} \quad (7.7)$$

When we apply this, the feedback we get from the original system (6.1) linear system  $\dot{y}_1 = v_1$  with transfer function:

$$F_1(s) = \frac{1}{s}$$

Now it is enough to design controller which will provide the required loop behavior. It will be necessary for it to have opened loop transfer function  $\frac{1}{T_1 s}$ . When we talk about simple first degree integrator, in this case it is enough use simple P-controller. Its gain will be

$$P_1 = \frac{1}{T_1}$$

For the two-level system (6.2) we will do the same. By requiring

$$\begin{aligned}\dot{y}_1 &= v_1 \\ \dot{y}_2 &= v_2\end{aligned}$$

the solution will be given equations for  $u_1$  and  $u_2$

$$\begin{aligned}u_1 &= A_1 v_1 + A_1 c_{12} \operatorname{sign}(x_1 - x_2) \sqrt{|x_1 - x_2|} + A_1 c_1 \sqrt{x_1} \\ u_2 &= A_2 v_2 - A_2 c_{12} \operatorname{sign}(x_1 - x_2) \sqrt{|x_1 - x_2|} + A_2 c_2 \sqrt{x_2}\end{aligned}$$

Similarly to previous case, after application of these feedbacks we obtain two independent (decoupled) systems  $\dot{y}_1 = v_1$ ,  $\dot{y}_2 = v_2$  having the transfer functions

$$\begin{aligned}F_1(s) &= \frac{1}{s} \\ F_2(s) &= \frac{1}{s}\end{aligned}$$

We can continue again in the same way as with the one level system. As we now have two independent linear systems, for each level we can design separate P-controller.

$$\begin{aligned}P_1 &= \frac{1}{T_1} \\ P_2 &= \frac{1}{T_2}\end{aligned}$$

We would like to recall as well the special case of two tanks connection. It differs from (6.1) in the way that the outflow from the first tank is at zero – valve 1 is closed, the second pump is turned off and the liquid's height in the second tank is controller by controlling the inflow into the first tank. We can write down the system equations as it follows:

$$\begin{aligned}
\dot{x}_1 &= \frac{1}{A_1}u_1 - c_{12}\sqrt{x_1 - x_2} \\
\dot{x}_2 &= c_{12}\sqrt{x_1 - x_2} - c_2\sqrt{x_2} \\
y &= x_2
\end{aligned} \tag{7.8}$$

The above-mentioned system has the relative degree  $r = 2$ . That's why we have to proceed, when looking for the feedback, from the equation:

$$\ddot{y} = v_1$$

so,  $u_1$  can be defined as

$$u_1 = \frac{2A_1}{c_{12}}v_1\sqrt{x_1 - x_2} + A_1c_{12}\sqrt{x_1 - x_2} - 2A_1c_2\sqrt{x_2} + A_1c_2\frac{x_1}{\sqrt{x_2}}$$

Different from the previous examples, by application of this feedback we obtain from the original system (7.8), linear system  $\ddot{y} = v_1$  with transfer function

$$F(s) = \frac{1}{s^2}$$

Thus, we would have to apply a different approach to continuous gainscheduling of the controller parameters based on (7.9), whereby the new input  $v_1$  should be computed in a way respecting all basic performance limitations – nonmodelled dynamics, model uncertainty, measurement and quantization noise and the control signal constraints.

## 7.5 PD<sub>2</sub> controller

In reality, the control signal is always constrained, what can be expressed as

$$u_r \in [U_1, U_2]; \quad U_1 < 0 < U_2 \tag{7.9}$$

So, despite to the fact that the control signal is usually generated by linear controllers, the input to the plant  $u_r$  is modified by a nonlinear function that can be denoted as the *saturation* function

$$u_r(t) = \text{sat}\{u(t)\}; \quad \text{sat}\{u\} = \begin{cases} U_2 & u > U_2 > 0 \\ u & U_1 \leq u \leq U_2 \\ U_1 & u < U_1 < 0 \end{cases} \tag{7.10}$$

In the classical period of control (up to late 60s in the 20th century), the constraints in the control were treated by a huge amount of papers. Then, in the subsequent decades in the main stream of the control theory

the problem of constraints practically disappeared. Just a limited number of authors dealing with the anti-windup design continued to investigate this important feature. Today, the problem of constraints is again in the focus of the research activities and it is hardly possible to give here just a brief overview of different approaches.

For the sake of brevity, we will show here briefly one possible approach to controlling the coupled tanks combining constrained pole assignment control [Huba and Bisták \(1999\)](#); [Huba \(1999\)](#); [Huba et al \(1999\)](#); [Huba \(2001, 2005, 2006, 2010, 2011a,c\)](#) with extended exact linearization method that is based on transformation of the existing nonlinear system by (7.9) into the double integrator one and in controlling such double integrator system by the PD<sub>2</sub> controller that fully considers the existing control saturation limits.

Let us start with considering the double integrator system

$$\frac{d^2\bar{y}(t)}{dt^2} = K_s\bar{u}(t) \quad (7.11)$$

For  $y = \bar{y} - w$ ;  $w$  being the reference input,  $\mathbf{x}(t) = [y(t), d(t)]$ ;  $d = dy/dt$  the system state and

$$u = K_s\bar{u}; \quad U_i = K_s\bar{U}_i; \quad i = 1, 2 \quad (7.12)$$

being the normalized control, it can be described in the state space as

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{A}\mathbf{x}(t) + \mathbf{b}u(t); \quad \mathbf{A} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}; \quad \mathbf{b} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad (7.13)$$

The linear pole assignment PD-controller is given as

$$u = \mathbf{r}^t \mathbf{x}; \quad \mathbf{r}^t = [r_0; r_1]; \quad \bar{u} = u/K_s \quad (7.14)$$

The PD<sub>2</sub> controller is composed from the linear pole assignment controller used for the relatively low velocities and given as

$$u = \frac{\mathbf{a}^t [\alpha_2 \mathbf{I} - \mathbf{A}]}{\mathbf{a}^t \mathbf{b}} \mathbf{x} \quad (7.15)$$

$$r_0 = -\alpha_1 \alpha_2; \quad r_1 = \alpha_1 + \alpha_2 \quad (7.16)$$

For  $d \in [d_0^2, d_0^1]$  the corresponding constrained pole assignment controller is again given by (7.15), (7.16) and with limiter (7.10). For higher velocities  $d$  the nonlinear algorithm is used that moves the representative point in the state space towards the so called Reference Braking Trajectory and guaranteeing braking with dynamics specified by one of the closed loop poles and without overshooting. The controller for the double integrator is then given as

$$u = \left[ 1 - \alpha_2 \frac{y - y_b}{d} \right] U_j \quad (7.17)$$

$$\bar{u}_r = \text{sat} \{u/K_s\} \quad (7.18)$$

$$j = (3 + \text{sign}(y)) / 2 \quad (7.19)$$

When controlling the nonlinear system (7.9), it is at first necessary to transform by means of inverse equation to (7.9) the existing saturation limits (e.g. 0 and 5 V) into the limits valid for the fictive controller input  $v_1$ . Then, using information about the plant state and the new saturation limits considered in (7.19), the fictive control signal  $v_1$  is computed by means of (7.15), or (7.19) and by using (7.9) the real control signal is achieved.

When substituting for  $x_1, x_2$  in the above transformations directly the actual level values, we use standard approach of the exact linearization. Due to the non-modeled dynamics it is, however, better to work with generalized operating points for  $x_1, x_2$  defined somewhere between the actual and the final state as

$$x_i = w_i * m_i + (1 - m_i) h_i; m_i \in \langle 0, 1 \rangle; i = 1, 2 \quad (7.20)$$

In this way we are using a method that represents combination of the exact linearization and of linearization around fixed operation point.

Example of transient responses are in Fig. 7.5 and Fig. 7.6. The traditionally used tuning derived by the conditions of the triple real pole

$$\alpha_{1,2} = -0.321/T_d \quad (7.21)$$

used in older works, the closed loop poles were now chosen according to Huba (2011b) as

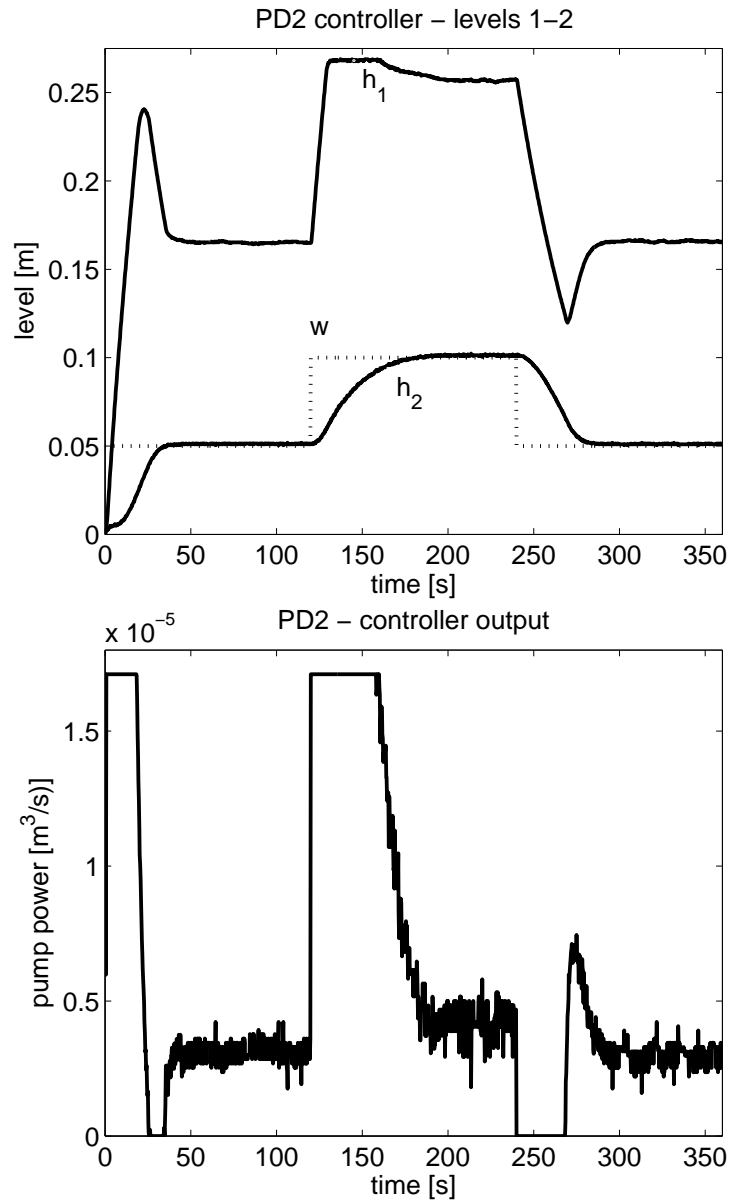
$$\alpha_{1,2} = -0.321/T_d \quad (7.22)$$

where  $T_d = 0.8s$  is the identified loop dead time, was now confronted with newer one Huba (2011b) derived by the performance portrait method. The dynamics of the control signal changes is obviously from the dynamical class 2. But, due to the plant character the second pulse in the control transients is not so dominant as the first one, what is a typical feature of all stable systems.

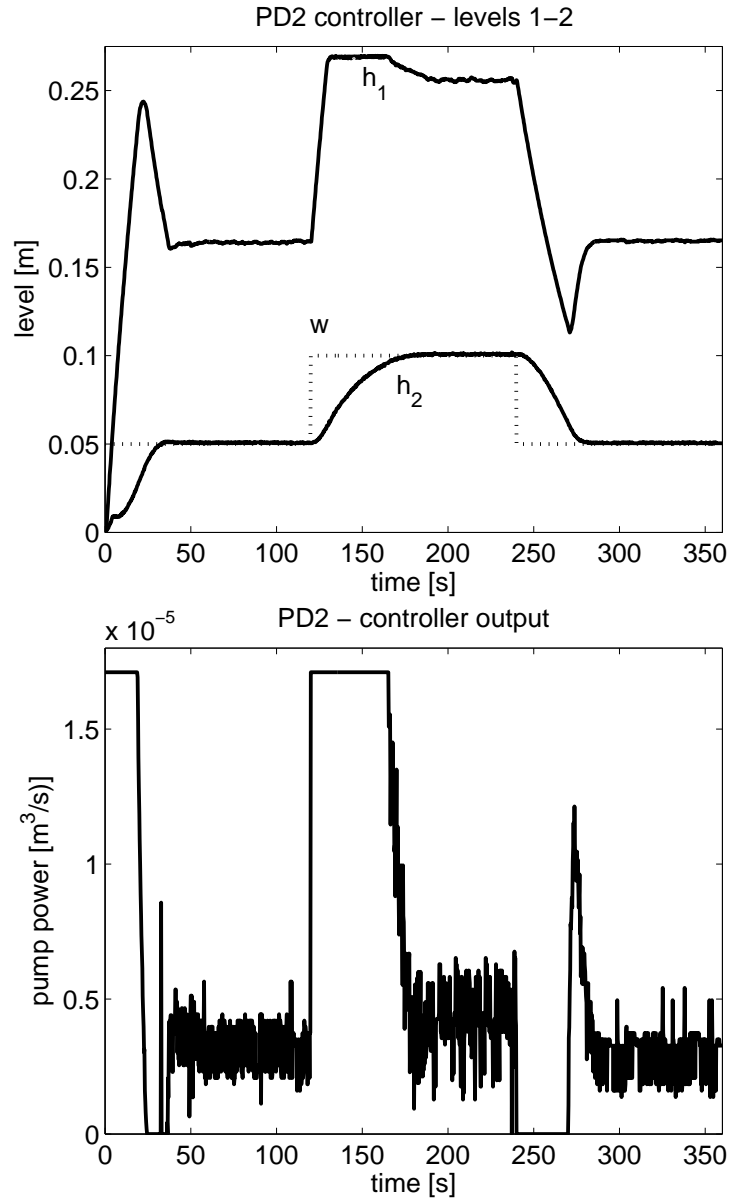
The controller structure in Simulink is depicted in Fig. 7.7. Control program which works with results of automatic calibration and identification is listed below.

### 7.5.0.2 Program for Matlab/Simulink

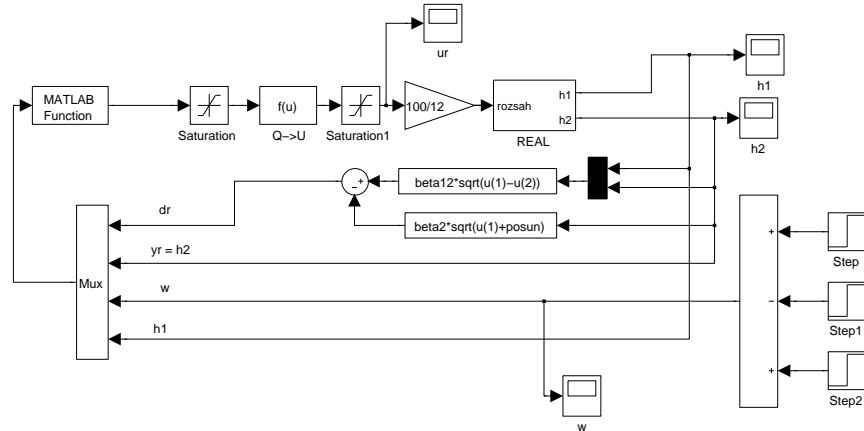
```
clc;
clear all;
close all;
```



**Fig. 7.5** Typical setpoint steps in the loop with PD<sub>2</sub> controller with tuning (7.21) may contain up to two step at saturation limits after larger setpoint steps at its output, but for stable systems the 2nd interval at saturation corresponding to braking the transient may disappear or not reach the saturation



**Fig. 7.6** Typical setpoint steps in the loop with PD<sub>2</sub> controller with tuning (7.22) derived by the performance portrait method Huba (2011b) gives a bit tighter control and faster dynamics of transient responses than the traditionally used tuning (7.21) what may be e.g. observed by longer period at the maximal level of  $h_1$  in the first tank and yet more by higher noise amplification



**Fig. 7.7** Control structure for nonlinear PD1 implemented in Matlab / Simulink. Matlab function block call PD2( $u(1), u(2), u(3), u(4)$ )

```

global alfa1 alfa2 Q1min Q1max m1c m2c beta1 beta12 beta2 posun
S1
% alfa1, alfa2 = closed loop poles
% Q1min Q1max = limit values for the pump input
% m1c,m2c = waighting coefficients for specifying operating
    points for
% linearization
% m1c,m2c=0 corresponds to exact linearization, i.e.
    linearization around
% actual state - leading to overshooting thanks to nonmodelled
    dynamics
% m1c,m2c=1 corresponds to linearization around reference state
    - too slow
% closed loop dynamics
% recommended to work with m1c,m2c>0 (0.1-0.2)
% beta1 beta12 beta2 = valve coefficients (just beta12 % beta2
    required)
% posun = offset of the output valve (difference between tank
    bottom and
% valve output orifice
% S1 = tank area

load '../basics/calibParams';
load '../basics/valve1';
load '../basics/valve2';
load '../basics/valve12';

```



```

load '.././basics/pump1';

T = 0.25;
           % sampling period

S1=0.001;    % tank area
posun = posun2; % offset 2

m1c=0.12; m2c=0.12;

beta1=0;beta12=b12;beta2=c2; % valves flow coeff.

Q1min=0;
Q1max=q_max1;

w11=0.05;
w12=0.10;
w13=0.05;
% setpoint values for tank 2 level - 3 steps
D=0.996;
% considered dead time
c=1;
alfae=-0.321/(c*D);
% closed loop pole choice c>1 is used for PID2 controller
    tuning

Tfil=-1/alfae;
Tfd=Tfil/10;
% Time constants for Disturbance Observer filter and derivative
    filter

alfa1=alfae;alfa2=alfa1;
% tuning for controller with real poles

sim('PD2_REAL_zilka.mdl');
str_datum = num2str(floor(clock));
str_datum = (strrep(str_datum, '____', '-'));
file_name = sprintf('reg_PD2-%s', str_datum);
eval(['save_', file_name]);

    Finally, also PD2 algorithm (PD2.m).

function out = PD2(dr,yr,w,h1m);
% constrained pole assignment controller for double integrator
    plant
% dr = dh2/dt - estimated according to the first state equation

```

```

% yr = h2    - measured level in tank 2
% w = reference value for tank 2
% h1m=h1     - measured level in tank 1

global alfa1 alfa2 Q1min Q1max m1c m2c beta1 beta12 beta2 posun
      S1
% alfa1, alfa2 = closed loop poles
% Q1min Q1max = limit values for the pump input
% beta1 beta12 beta2 = valve coefficients (just beta1 % beta12
      required)
% posun = offset of the output valve (difference between tank
      bottom and
% valve output orifice

% *****signals transformations*****
h2m=yr; y = yr-w;
% s shifting reference state to the origin

w1 = ((beta2/beta12)^2)*(w+posun)+w;
% steady state level in tank 1
% corresponding to setpoint reference level w in tank 2
% posun = valve orifice offset

h1=m1c*w1+(1-m1c)*h1m;
h2=m2c*w+(1-m2c)*h2m;
% h1,h2 =operating points for linearization of tank 1 and 2
% m1c,m2c = waighting coefficients for specifying
% operating points for linearization
% m1c,m2c=0 correspond to exact linearization,
% i.e. linearization around actual state - leading
% to overshooting thanks to nonmodelled dynamics
% m1c,m2c=0 correspond to linearization around reference state
-
% too slow closed loop dynamics
% recommended to work with m1c,m2c>0 (0.1-0.2)

h12=abs(h1-h2);
z=sign(h1-h2);
a = beta12/(2*S1*sqrt(h12));
b = -(-beta12^2-1/2*(-2*beta2*h2-beta2*posun+beta2*h1)*beta12
      /((h1-h2)^(1/2)*(h2+posun)^(1/2))+1/2*beta2^2);
% transformation to the normal form for output y=h2
% real control ur of pump with constraints Q1min, Q1max
% replaced by fictive input of a double integrator
% with limits Q12 and Q11

```

```

Q12 = a*Q1max-b;
Q11 = a*Q1min-b;
% limit values of fictive input signal of double integrator

*****control algorithm*****
% control algorithm for constrained control
% of double integrators with real
% closed loop poles alfa1, alfa2
if (Q11*Q12>0)
    if (h1m+h2m>w+w1)
        u=Q1min;
    else
        u=Q1max;
    end
    ur=u;
else
    if y>0
        Q1j = Q12;
    else
        Q1j = Q11;
    end

    d1=Q11/alfa1;
    d2=Q12/alfa1;

    if(dr>d2) & (dr<d1)
        u=(-alfa1*alfa2*y+(alfa1+alfa2)*dr);
        % linear pole assignment
    else
        u=((1-alfa2*(y-0.5*(dr*dr/(Q1j)+Q1j/(alfa1*alfa1)))/dr)*
            Q1j);
        % nonlinear pole assignment
    end

    % constraining output
    if (u>Q12)
        u=Q12;
    end

    if (u<Q11)
        u=Q11;
    end
end
% *****inverse transformation*****
% inverse transformation from the fictive

```

```
% double integrator input u to
% real input ur
ur = (u+b)/a;
end

out = ur;
```

## 7.6 Conclusion

We have described the basic possibilities of control of available hydraulic system established by different configuration of one or two containers with one or two pumps. It is important to mention, that in case of control of real system with the mentioned controllers we will not obtain the ideally supposed results. Certainly, due to modelling errors we will register some deviations between supposed and real transients. The control performance will depend on the accuracy of identification and robustness of controller. As we will never be able to define the system's parameters with 100% precision, the system will never behave as the perfect integrator and P-controller will not assure the necessary quality of control. That's why we propose to design the robust controllers having sufficient robustness against influence of disturbances, uncertainties and measurement noise. At the same time, these controllers should also respect the available system's limits. Such proposals for one-level system can be found for example [Huba \(2003\)](#); [Pan et al \(2005\)](#); [Halás \(2006\)](#); [Almutairi and Zribi \(2006\)](#); [Žilka et al \(2009\)](#). Further development of the above-mentioned procedures, as e.g. the disturbance decoupling problem of coupled tanks was studied in [Žilka and Halás \(2010\)](#) and non-interacting control of coupled tanks in [Žilka and Halás \(2011\)](#).

**Acknowledgements** The work has been partially supported by the Slovak Grant Agency grants No. VG-1/0656/09 and VG-1/0369/10. It was also supported by a grant (No. NIL-I-007-d) from Iceland, Liechtenstein and Norway through the EEA Financial Mechanism and the Norwegian Financial Mechanism. This project is also co-financed from the state budget of the Slovak Republic.

## References

- Almutairi NB, Zribi M (2006) Sliding mode control of coupled tanks. *Mechatronics* 16(7):427 – 441
- Conte G, Moog C, Perdon A (2007) Algebraic Methods for Nonlinear Control Systems. Theory and Applications, 2nd edn. Communications and Control Engineering, Springer-Verlag, London
- Halás M (2006) Quotients of noncommutative polynomials in nonlinear control systems. In: 18th European Meeting on Cybernetics and Systems Research, Vienna, Austria

- Huba M (1999) Dynamical classes in the minimum time pole assignment control. In: Computing Anticipatory Systems - CASYS '98, Woodbury: American Institute of Physics, pp 499–512
- Huba M (2001) Constrained control of the two tank system. In: 9th Mediterranean Conference on Control and Automation, Dubrovnik, pp 93–98
- Huba M (2003) Gain scheduled *PI* level control of a tank with variable cross section. In: 2nd IFAC Conference on Control Systems Design, Bratislava, Slovakia
- Huba M (2005) P- und pd-polvorgaberegler für regelstrecken mit begrenzter stellgröße. *Automatisierungstechnik AT* 53(6):273–283
- Huba M (2006) Constrained pole assignment control. In: Current Trends in Nonlinear Systems and Control, L. Menini, L. Zaccarian, Ch. T. Abdallah, Edts., Birkhäuser, Boston, pp 163–183
- Huba M (2010) Designing robust controller tuning for dead time systems. In: IFAC Int. Conf. System Structure and Control, Ancona, Italy
- Huba M (2011a) Basic notions of robust constrained pid control. In: Selected topics on constrained and nonlinear control. M. Huba, S. Skogestad, M. Fikar, M. Hovd, T.A. Johansen, B. Rohač-Ilkiv Editors, STU Bratislava - NTNU Trondheim
- Huba M (2011b) Constrained pole assignment controller for sopdt plant. complex poles. In: NIL Workshop on constrained and nonlinear control. M. Huba, S. Skogestad, M. Fikar, M. Hovd, T.A. Johansen, B. Rohač-Ilkiv Editors, STU Bratislava - NTNU Trondheim
- Huba M (2011c) Robust controller tuning for constrained double integrator. In: NIL Workshop on constrained and nonlinear control. M. Huba, S. Skogestad, M. Fikar, M. Hovd, T.A. Johansen, B. Rohač-Ilkiv Editors, STU Bratislava - NTNU Trondheim
- Huba M, Bisták P (1999) Dynamic classes in the pid control. In: Proceedings of the 1999 American Control Conference, San Diego: AACC
- Huba M, Sovišová D, Oravec I (1999) Invariant sets based concept of the pole assignment control. In: European Control Conference ECC'99, Duesseldorf: VDI/VDE
- Pan H, Wong H, Kapila V, de Queiroz MS (2005) Experimental validation of a nonlinear backstepping liquid level controller for a state coupled two tank system. *Control Engineering Practice* 13(1):27 – 40
- Žilka V, Halás M (2010) Disturbance decoupling of coupled tanks: From theory to practice. In: IFAC Symposium on System, Structure and Control - CSSS 2010, Ancona, Italy
- Žilka V, Halás M (2011) Noninteracting control of coupled tanks: From theory to practice. In: 13th International Conference on Computer Aided Systems Theory - EUROCAST 2011, Las Palmas, Gran Canaria
- Žilka V, Halás M, Huba M (2009) Nonlinear controllers for a fluid tank system. In: R. Moreno-Díaz, F. Pichler, A. Quesada-Arencibia (Eds.): Computer Aided Systems Theory - EUROCAST 2009, Springer, Berlin, Germany, Lecture Notes in Computer Science, pp 618–625

*Comments – Remarks*

*Comments – Remarks*

*Comments – Remarks*



*Comments – Remarks*

# Chapter 8

## Remote Laboratory Software Module for Thermo Optical Plant

Pavol Bisták

**Abstract** Virtual and remote laboratories play an important role in the education process of engineers. Their expansion is closely connected to the growth of Internet. This contribution describes how it is possible to install and use the remote laboratory for the thermo optical plant. It explains the process of installation both the server and the client applications. Then the user interface of the client application is discussed in details. An administrator of the remote laboratory can modify existing control structures therefore there are special rules included in one section how to create new Simulink block diagrams in order to be compatible with the remote laboratory software. Thus one can set up the remote laboratory easily and it can be modified to her/his needs quickly.

### 8.1 Introduction

The remote laboratory software module for the uDAQ28/LT thermo optical plant is represented by the client server application which main task is to enable remote experiments via Internet. Although the software has been designed to co-operate with the uDAQ28/LT real plant its usage is not limited only to this real system. The server part of the software is created by the program application written in the Java programming language. The server application should be installed on the computer that directly controls the real plant. The client part of the software is realized in the form of a Web page with embedded Java applet. This provides a user friendly environment to carry out remote experiments.

---

Pavol Bisták  
Institute of Control and Industrial Informatics, Slovak University of Technology in Bratislava, e-mail: [pavol.bistak@stuba.sk](mailto:pavol.bistak@stuba.sk)

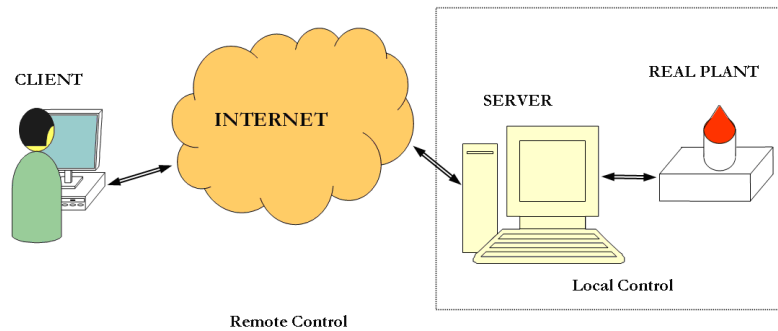


Fig. 8.1 Remote laboratory overview

## 8.2 Technical Requirements

### 8.2.1 Server

- Windows operating system (XP, Vista, Seven)
- Java Virtual Machine
- Matlab with Simulink
- fixed IP address
- free port No. 23

### 8.2.2 Client Computer

- Internet browser with Java applet support

## 8.3 Installation

### 8.3.1 Server Installation

First it is necessary to check whether a *Java Run Time (JRE)* environment is installed on the server computer. By default Java is installed into the *Program Files* folder. To check the Java installation you can also open the Command Window (*Start/Run/cmd*) and type '*java*' command to see whether the Java installation exists or no. If there is no Java installed on the server computer you should download the installation file from the *www.javasoft.com* and install it.

The proper installation of the uDAQ28/LT thermo optical plant is another preposition. The installation should be carried out according to the guide on the installation CD. To check if the control of the real plant works locally it is possible to run one of the predefined experiments from the installation CD.

In the case the above mentioned requirements are fulfilled the server application could be installed. Open the *Server* folder on the installation CD and double click the *server\_install.bat* file. During the process of installation there will be a new folder called *RemLabWorkspace* created in the root directory of the *C:* drive. Necessary Matlab files will be copied into this folder. Moreover the server application *RemLabServer.exe* will be copied into the *RemLabWorkspace* folder. Before running the server application it is recommended to check the setting of the serial port (in supplied block diagrams that are placed in the directory *C:\RemLabWorkspace*).

**Warning:** Do not delete or move the folder *C:\RemLabWorkspace*. This folder is fixed working directory of the server application. The *RemLabServer.exe* application file can be moved from the *C:\RemLabWorkspace* folder to any different place and it is also possible to create a shortcut placed at the desktop in order to run it quickly.

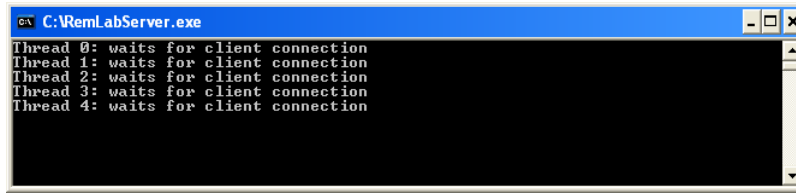
### 8.3.2 Client Installation

The client application is created in the form of a Java applet inserted into a Web page. For the testing purposes the installation is not needed because it can be run directly from the installation CD by clicking the *client.html* file placed in the *Client\Application* folder.

But the client application is aimed for a Web publication when it can be accessible for all potential users. In this case it is necessary to upload the content of the *Client\Application* folder to the desired Web server. Then after setting the correct Internet address the client application is downloaded into the Web browser window that enables running of Java applets.

## 8.4 Running the Client Server Application

The run of a client server application must be synchronized. First it is necessary to run the server part of the application. After the server has been successfully installed it is possible to run it by a double click to the *RemLabServer.exe* file that is (after the correct installation) placed in the *C:\RemLabWorkspace* folder. Depending on the server computer safety settings it can happen that the system will ask for confirmation of a trust to the application.



```

C:\RemLabServer.exe
Thread 0: waits for client connection
Thread 1: waits for client connection
Thread 2: waits for client connection
Thread 3: waits for client connection
Thread 4: waits for client connection

```

Fig. 8.2 Server application waiting for client connection

It can be noticed that the run of the server application invokes the run of Matlab application automatically. After starting the server is held in the state of waiting for the connection from the client. After the client connects to the server it starts to fulfill client's demands. The server usually runs without breaking (if the safety issues are covered). In the case it is necessary to finish the server application, it can be done by closing the *RemLabServer.exe* application window and closing manually the Matlab application.

If the server starts successfully it is possible to connect to it from the client application. To run the client application means simply to open the starting Web page *client.html* placed in the *Client\Application* folder. Usually the client application is installed on a Web server. Then it is necessary to set the corresponding Web page address into the Web browser. Depending on the Web browser safety settings it could be required to confirm a trust in running the Java applets.

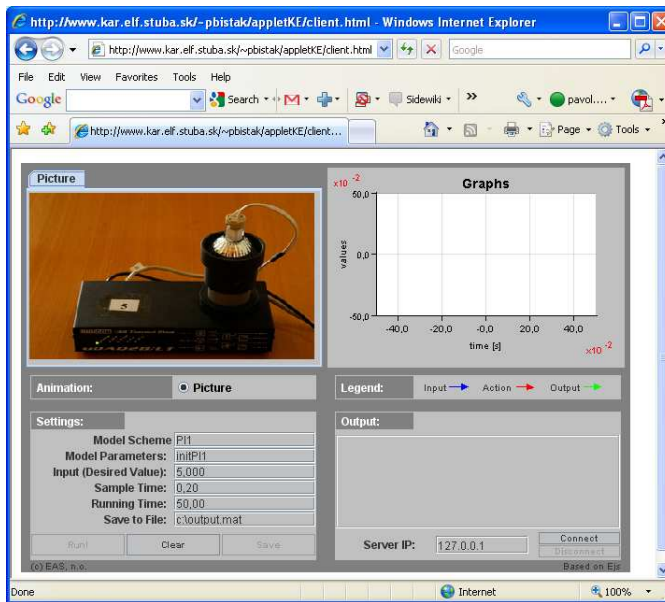


Fig. 8.3 Start of the client application after downloading Java applets

## 8.5 Client User Interface

The client application serves as the control panel for remote experiments. Its interface must provide remote users with tools that will enable to exploit all functions of the server side. Generally this interface allows its users to connect to the server, set parameters of experiments, control the run of experiments, visualize experiments in the form of animations, graphs, numerical data and finally to save data for later processing.

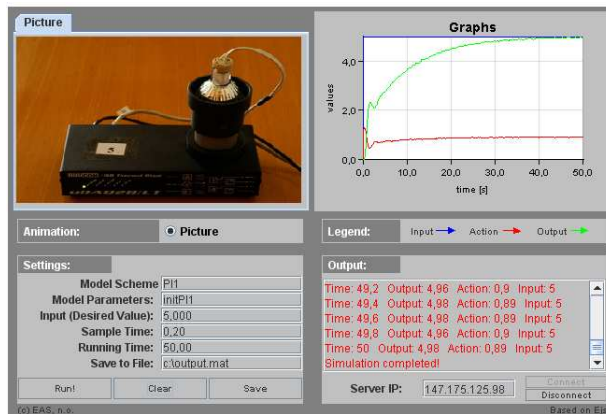


Fig. 8.4 Graphical user interface of client application

The graphical user interface of the client application can be divided to four quadrants. The two upper quadrants cover a graphical visualization of data from the running experiment. In the left upper quadrant the experiment is animated by the series of pictures selected from a database according to the current state values of the experiment.

In the upper right quadrant there are experiment data displayed in the form of graphs. There are graphs of input (desired), output and control values. These responses give an overview how the experiments evolves. More detailed responses can be acquired after saving the numerical data and displaying them using different software for graphs (Matlab, e.g.). In the displayed graphs the scale of the axis is continuously modified according to the actual values of the displayed data. Below the graphs there is a *Legend* for individual responses. When the rules for creation of block diagrams (Section 8.7) will be kept the blue curve will represent the input signal, the red curve will be the control value and the green one is for the output value of the real system.

The two lower quadrants can be characterized as alphanumeric ones because they serve for input and display of textual and numerical data of ex-

periments. Beside this there is a group of buttons placed in the bottom part of the interface that enables to control the whole client application.

The lower left sector *Settings* assures the setting of the following experiment parameters: the name of the controlled model, model parameters, input (desired) value, sampling period, duration of experiment and the name of a data file to which the state variables of the running experiment can be saved.

The lower right sector *Output* displays the experiment outputs in the numerical form. If the user does not want to exploit the possibility to save data into the data file it is possible to copy data from the field *Output* through the clipboard to arbitrary program for further processing. Besides this the field *Output* displays also the application state messages (whether the connection has been successful, if no then why, the state of saving data to the external file, etc. ). Further it shows warnings and error messages in the case they are generated by the system.

The buttons for the control of running the application are placed at the bottom part of the interface and play the role of a connection establishment or cancellation, running the experiment, saving the experiment data, clearing the interface before running a new experiment. Besides the control buttons there is also the textual field in the bottom part to which it is necessary to enter the IP address before the connection is established.

### 8.5.1 Settings

The following parameters can be set up using the lower left sector *Settings*:

- *Model Scheme* – the name of the Simulink file (.mdl) that represents the block diagram model of the experiment. This file must be placed on the server computer in the specified folder *C:\RemLabWorkspace* that was created during the server installation. The block diagram must keep the rules written in Section 8.7. The user should be informed about the existing names of .mdl files that are available.
- *Model Parameters* – the name of the parameter file that is represented by Matlab .m file. Also this file must be placed on the server computer in the folder *C:\RemLabWorkspace*. Usually it contains the values of constants and calculations of initial parameters necessary for running the experiment (block diagram model).
- *Input(Desired Value)* – input value of the block diagram that in the case of controller represents a desired value of the output value. Because the real system has limits included in the block diagram it has only sense to set the desired value from the certain interval of values.
- *Sample Time* – sampling period. Model state variables are measured and changed only in the regular instants characterized by this value. The minimal sampling period is given by physical possibilities of the system.

- *Running Time* – duration of the experiment. It limits the time how long a selected block diagram with specified parameters will be running. During this time the model state variables are visualized by the help of graphs and animations. Also the numerical values of the model state variables are displayed in the field *Output*.
- *Save to File* – the name of the file placed on the user computer to which time sequences of the model state variables will be saved after pressing the button *Save*. It is necessary to enter the whole path (C:\output.mat, e.g.). If the file does not exist it will be created.

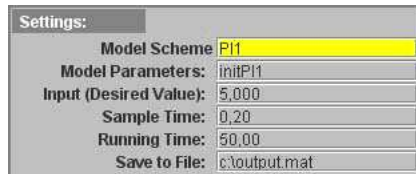


Fig. 8.5 Settings

**Warning:** When entering any parameter the corresponding field is highlighted by yellow color that indicates data editing. The new parameter is valid only if the editing is finished by pressing the *Enter* key.

### 8.5.2 Server IP Address and Control Buttons



Fig. 8.6 Control buttons

There are a field for setting the server IP address and following control buttons in the bottom part of the client interface:

- *Server IP* – the field for setting the server IP address in the form of four three-digit numbers separated by points. The IP address should be fixed and the user should know it. For testing purposed it is possible to start the client application on the server computer. In this case it is necessary to enter the value *127.0.0.1* or the string *localhost*.
- *Connect* – this button realize the connection to the server. A user is informed about the result of the connection in the field *Output*. When the connection is successful the message is *Connected to server....* In the opposite case (wrong IP setting, e.g.) the message is *Could not connect to*



*remote host* (some time it takes a longer period). In the case when someone is already connected to the server, the connection with another client will be not realized because in one time only one user can operate a real hardware. This time the message is: *Server busy – disconnecting*



**Fig. 8.7** Connection

- *Disconnect* – the button that cancels the connection with the server and thus the server is free to accept connection from another user. There is a message *Disconnected* in the field *Output* when the client is disconnected. If a client is not active for longer period or because of network failure she/he could be disconnected from the server automatically. She/he is informed by the message *Connection timeout or network error occurred. Disconnected from server.....*



**Fig. 8.8** Automatic disconnection

- *Run* – it serves for running the block diagram specified in the *Model Scheme* parameter. First all parameters are sent to the server where also the parameters from the file given in the *Model Parameters* field are read into the Matlab and consequently the experiment starts with continuous data transfer and visualization that takes until the time set in the *Running Time* field is over. In the *Output* field the message *Simulation starts...* will appear and the transfer data characterizing the experiment states begin to be displayed. The button is available only after a successful connection to the server.
- *Clear* – its task is to clear display fields of the client application, namely the *Output* and *Graphs* sectors. The application will be then prepared for the next run within the existing connection to the server.
- *Save* – this button is available after the duration (*Running Time*) of the experiment is over. When it is activated the time sequences of the model

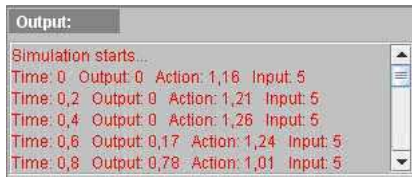


Fig. 8.9 Experiment run

state variables coming from the just finished experiment begin to save into the file specified by the *Save to File* parameter. In the *Output* field the user is informed about the process and success of saving data. The format of the saved file is identical with the Matlab workspace data format (.mat file) and therefore it possible to read data into the Matlab environment later on and process the saved data more carefully.



Fig. 8.10 Saving data into the file

## 8.6 Running the Experiment

The sequence of running the experiment is following:

1. The basic assumption is that the server is running. If it does not it is necessary to run it as it was written in Section 8.4.
2. Start the client application according to the guide in Section 8.4.
3. Set the IP address of the server into the *Server IP* field.
4. Press the *Connect* button and check if the message *Connected to server...* appeared in the *Output* field (Fig. 8.7).
5. Set the the parameters in the lower left sector *Settings*.
  - Select the block diagram and initial parameters by entering the names of corresponding files (*Model Scheme*, *Model Parameters*)
  - Set the desired value, sampling period and duration of the experiment
6. Press the *Run* button to start the operation of the selected block diagram. This will cause running the remote experiment. During the period of the

experiment running (specified in *Running Time*) it is not possible to interact with the experiment and it is necessary to wait until it finishes to take over the control of the application again. Depending on the network traffic and the load of the server it could happen that the experiment does not run correctly or the data are not transferred to the client. In this case the button *Run* must be pressed repeatedly.

7. When it is required that data from the just finished experiment have to be saved set the name of the data file into the *Save to File* field and press the *Save* button. It is necessary to wait for the end of the saving process that is announced in the *Output* field (Fig. 8.10). Then it is possible to process the data saved in the specified file using another application.
8. In the case of repeating the experiment with new parameters clear the *Output* and *Graphs* fields by pressing the *Clear* button and repeat this procedure starting from the point 5.
9. Press the button *Disconnect* when it is not desired to continue working with remote experiments. This will free resources of the remote laboratory for other users. To finish the client application close the corresponding window of the Internet browser.

## 8.7 Rules for Creation of Models in Simulink

Remote laboratory administrators can create new block diagrams and place them (together with initial parameter files) on the server in the fixed given folder *C:\RemLabWorkspace*. Thus remote users can have more possibilities to select the type of the experiment when choosing the corresponding names of files (.mdl and .m files) in the *Model Scheme* and *Model Parameters* fields. To assure the application is working correctly several simple rules must be kept when creating these new block diagrams.

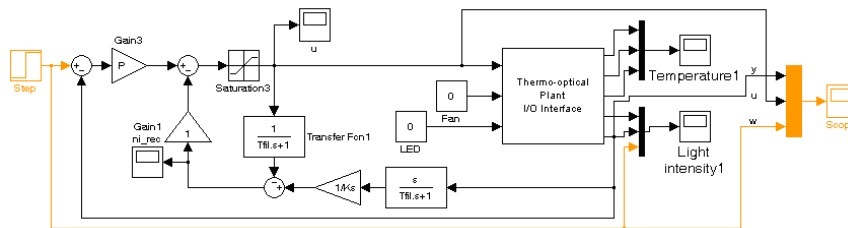
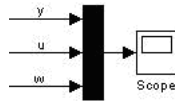


Fig. 8.11 Supplied block diagram

The block diagram depicted in the Fig 8.11 can be found in the *C:\RemLabWorkspace* folder after successful installation. The core of this block diagram

is created by the *Thermo-optical Plant I/O Interface* block that represents the physical model of the uDAQ28/LT thermo optical plant. The blocks drawn by the orange color are important for the correct operation of the remote laboratory software module. These are input/output blocks of this block diagram and the client application is programmed in that way that it could exchange the data with these blocks. The following three rules have to be kept where the first two are related to input parameters and the third one to input/output parameters of the block diagram:

1. The sampling period of an individual block as well as the whole block diagram is given by the global variable denoted as  $T_s$ . This denotation must be kept everywhere. If a variable with this name is used in the initial parameter file (.m file) or somewhere in the block diagram (.mdl file) its value will be overwritten by the value set in the *Sample Time* field of the client application.
2. The desired value is given by the global variable denoted as  $w$  (the *Final value* parameter of the *Step* block in the supplied block diagram, e.g.). This denotation must be kept everywhere. If a variable with this name is used in the initial parameter file (.m file) or somewhere in the block diagram (.mdl file) its value will be overwritten by the value set in the *Input(Desired Value)* field of the client application.
3. Signals that should be displayed in the client application interface have to be gathered using the multiplexer block into the only one *Scope* block whose name is identical with the name *Scope* and its settings coincide with those of Fig. 8.13.



**Fig. 8.12** Gathering signals into the Scope block

When creating new block diagrams it is recommended to start with the supplied block diagram (Fig. 8.11) and modify its structure while keeping the blocks drawn by the orange color unchanged. Thus it is possible to add an arbitrary controller to the thermo optical plant block that can be realized in the Simulink. The block diagram in the Fig. 8.11 and its initial parameter file have been developed by a simple modification of the original files (*PI1\_windupless.mdl* a *init\_PI1\_windupless.m*) that are supplied on the installation CD of the uDAQ28/LT thermo optical plant.

**Warning:** Time responses from the supplied block diagram and the parameter file should not be the optimal ones. First it is necessary to identify



**Fig. 8.13** Scope block parameter settings

each individual piece of the uDAQ28/LT thermo optical plant and consequently to modify parameters of the controller in the initial parameter file.

## 8.8 Conclusion

Remote laboratory software module extends possibilities how to exploit the thermo optical plant in that way that thank to the Internet it is accessible for greater number of interested persons. If the safety conditions are kept it can run 24 hours per day. Thus the time and distance barrier are decreased and the effectiveness of the real plant usage is enhanced.

To increase the accessibility of the remote laboratory it is recommended to include the starting Web page of the client application into a Learning Management System (LMS) or into other system for Web pages administration and provide the detailed description of the remote experiments that are available (introduce the file names of block diagrams and initial parameters that can be chosen). It is also advantageous to provide users with the value of the server IP address (if its value is not predefined in the client application).

The client application is universal so it can be easily modified to create a user interface for remote control of other real plants. This has been successfully tested for hydraulic plant, magnetic levitation system and rotational pendulum.

*Comments – Remarks*

*Comments – Remarks*

*Comments – Remarks*



*Comments – Remarks*

# Chapter 9

## Constrained PID control

### Tasks for Controlling the Thermo Optical Plant

Peter Ľapák and Mikuláš Huba

**Abstract** Tuning of simple controllers respecting input constraints will be considered in this chapter and verified by controlling simple physical device of thermo-optical plant. The chapter starts with short user's manual and installation guide to the uDAQ28/LT device which will be used as a real plant to apply the control on. In the introduction, several fundamental controllers of the Dynamical class 0 (DC0) will be considered that avoid control saturation by guaranteeing monotonic transients among steady states at the controller output. Processes of the DC0 are typically met in situations, where the dynamics of transients may be neglected, i.e. it is not connected with a reasonable energy accumulation. In such cases, the ideal control response following a setpoint step may also converge to step function (not having a saturation phase). Controllers of the DC0 may also be successfully applied to any stable plant, but in such situations it is no more possible to speed up the control signal transient up to the step function, just to keep it monotonic that guarantees that all such controllers may again be successfully treated by the linear theory as well. In the second part, basic structures of the DC1 will be introduced that may already typically have one constrained period in their control signal step responses. Here, control structures with integral action based on disturbance observers will be introduced that do not exhibit windup phenomenon and so enable simpler one-step tuning that in the case of traditional linear controllers extended by the anti-windup circuitry.

---

Peter Ľapák  
Institute of Control and Industrial Informatics  
Faculty of Electrical Engineering and IT, Slovak University of Technology  
Ilkovičova 3, 812 19 Bratislava, Slovakia, e-mail: [peter.tapak@stuba.sk](mailto:peter.tapak@stuba.sk)

Mikuláš Huba  
Institute of Control and Industrial Informatics  
Faculty of Electrical Engineering and IT, Slovak University of Technology  
Ilkovičova 3, 812 19 Bratislava, Slovakia, e-mail: [mikulas.huba@stuba.sk](mailto:mikulas.huba@stuba.sk)

## 9.1 Thermo-optical Plant uDAQ28/LT – Quick Start

This section gives a short guide to uDAQ28/LT plant. Get familiar with thermo-optical plant interface. For more information on the device please refer to the user's manual. This device offers measurement of eighth process variables (temperature and its filtered value, ambient temperature, light intensity, its filtered value and its derivative, the ventilator speed of rotation and its motor current). The temperature and the light intensity control channels are interconnected by three manipulated variables: the bulb voltage (the heat and light source), the light-diode voltage (the light source) and the ventilator voltage (the system cooling). The plant can be easily connected to standard computers via USB, when it enables to work with the sampling periods 40-50 ms and larger. Within the Matlab/Simulink scheme the plant is represented as a single block, limiting use of costly and complicated software package for the real time control.

### 9.1.1 Installation in Windows Operating System

#### 9.1.1.1 Device Driver Installation

New hardware is detected automatically by operating system and *Found New Hardware Wizard* will start after plugging device into electrical power network and connecting it to PC by USB cable (Fig. 9.1). Choose driver installation from specific location (Fig. 9.2)

Directory *driver* from the package of supporting software is needed to select as device driver location path (Fig. 9.3). Complete installation requires two cycles of adding new hardware (*USB Serial Converter* and *USB Serial Port* is installed into operating system, *Found New Hardware Wizard* starts automatically second time). The user is informed on successful completion of installation, see the window on the Fig. 9.4.

When the device has been installed, (virtual) serial port is available on the list of hardware devices, labeled as USB Serial Port. One can get to its settings through Device Manager (Start / Settings / Control Panel / System / Hardware / Device Manager ) if Ports (COM & LPT) / USB Serial Port is selected (Fig. 9.5). The port has automatically assigned by the system one of com port numbers (which is not in use). If the assigned number is bigger than 4, it is necessary to change it to different com port number from the range 1-4, which is not in use at the moment. It is possible to perform on the tab Port Settings / Advanced / COM Port Number (Fig. 9.6). The last action after installation to be done is setting parameter Latency Timer (msec) on value 1, which minimizes data loss during data transfer. Parameter Latency Timer (msec) is available on the same tab as COM Port Number (Fig. 9.6). In the case that all com port numbers from the range 1-4 are already in use by



Fig. 9.1 Found new hardware wizard

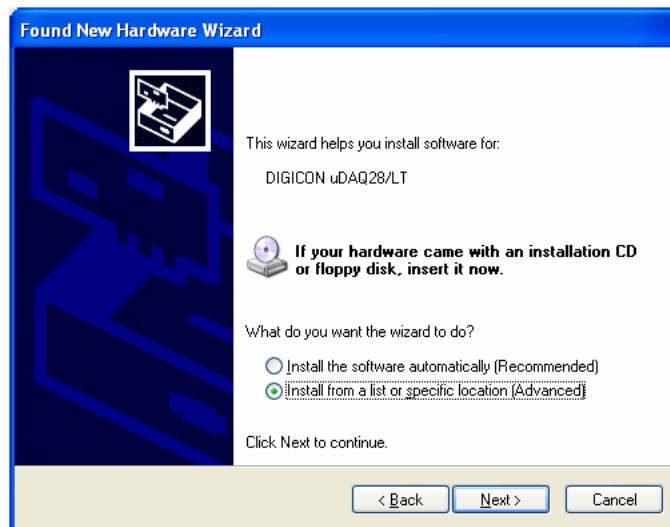


Fig. 9.2 Install driver from specific location selection

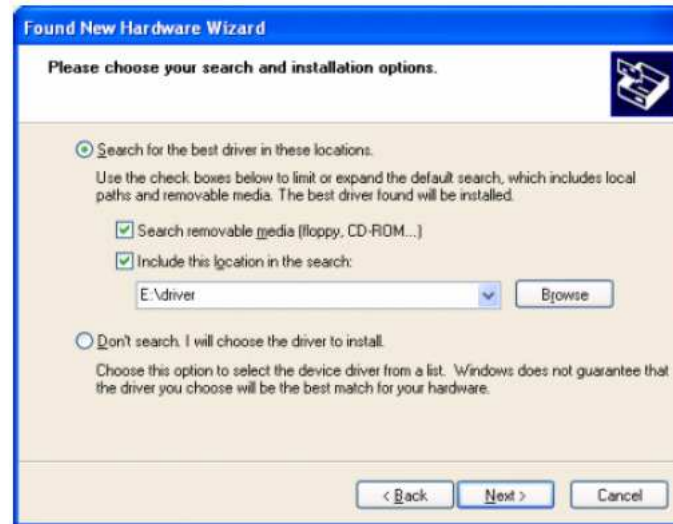


Fig. 9.3 Driver location selection



Fig. 9.4 Successful completion of installation

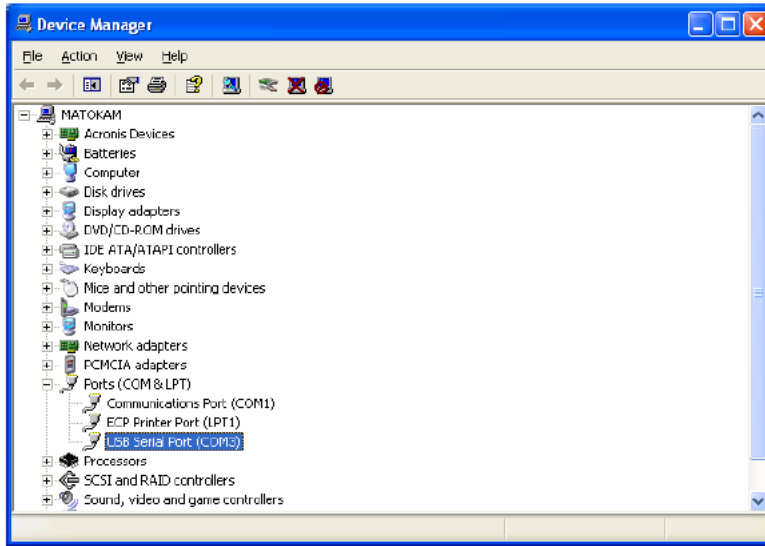


Fig. 9.5 Device manager – USB serial port

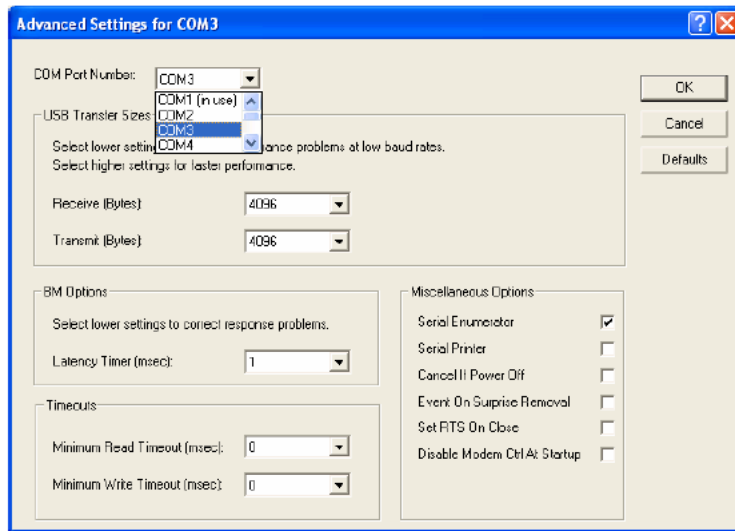


Fig. 9.6 Assignment COM port number to USB Serial Port, Latency Timer setting

other applications or programs, first assign (change) com port number bigger than 4 to one of those programs and then assign available number from 1 to 4 to USB Serial Port.

### 9.1.1.2 Driver and Software Package Installation in Matlab

To install driver and software package in Matlab, follow these instructions:

- Before installation it is good to make sure that any untermiated process `matlab.exe` is not in the memory; if it is, terminate it (tab Processes in Task Manager, activated by keys CTRL+ALT+DEL)
- Run Matlab
- Change the working directory to the directory from the package of supporting software where installation file `udaq_setup.p` is located (e.g. by typing command `cd e:\dirname` )
- Run file `udaq_setup.p` by typing command `'udaq_setup'`

After successful completion of installation there is directory `matlabroot/udaq/udaq28LT` and particular subdirectories and files copied on the local hard drive. (`matlabroot` represents the name of the directory displayed after typing command `'matlabroot'` in Matlab command window) Two mdl files are open after installation in Matlab: `udaq28LT_iov2.mdl`, located in `matlabroot/udaq/udaq28LT/examples` and the library containing two blocks (drivers), represented by mdl file `matlabroot/udaq/udaq28LT/blks/udaq28LT_lib.mdl`. You can create your own simulation mdl file that communicates with thermo-optical plant by copying the driver block from the library (or from other functioning mdl file) into your own mdl file.

### 9.1.1.3 Thermo-optical Plant Communication Interface

Thermo-optical plant communication interface is represented in Matlab by one of the blocks `udaq28LT_v1R13` (Fig. 9.7) or `udaq28LT_v2R13` located in the library `matlabroot/udaq/udaq28LT/blks/udaq28LT_lib.mdl`. Double clicking on the `udaq28LT_v1R13` block brings up the block parameters menu (Fig. 9.8).

### 9.1.1.4 Measurement and Communication System

The inputs and outputs of the communication interface refer to these signals.

Inputs:	Bulb	0-5 V to 0-20 W of light output
	Fan	0-5 V to 0-6000 fan rpm
	LED	0-5 V to 0-100% of LED light output
	T, D	microprocessor inputs for the purpose of calculation of the first light channel derivative (sample period the microprocessor samples light channel with – minimal possible value is 1 ms and coefficient of actual sample for the

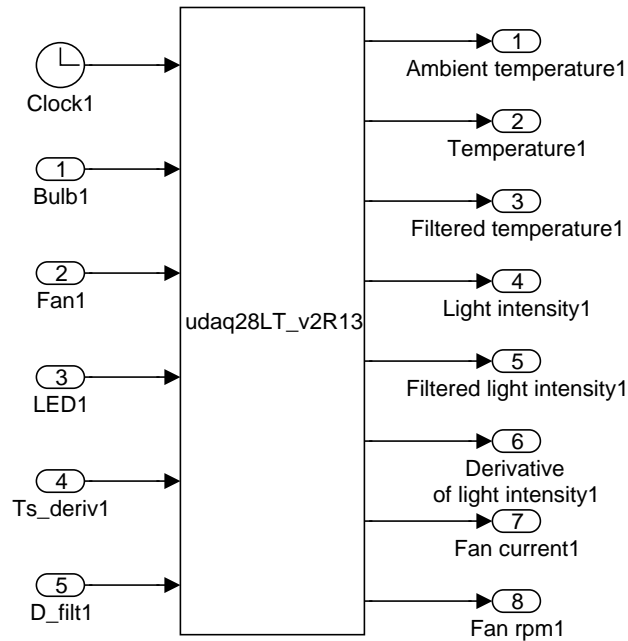


Fig. 9.7 Communication interface block in Simulink

discrete filter of the first order with accuracy of 3 decimal positions)

Outputs: Ambient temperature  
 Temperature sensor PT100  
 range 0 – 100 °C  
 accuracy: better than 99%  
 Filtered temperature (1st order filter with time constant cca 20 s)  
 Light intensity  
 Filtered light intensity (1st order filter with time constant cca 20 s)  
 Filtered derivative of the first light intensity channel  
 Current consumption by fan (0-50 mA)  
 Fan revolutions (0-6000 rpm)

Operating temperatures range: 0 – 70 °C  
 Power supply: 12V/2A DC (external adapter)  
 Communication interface: USB – virtual serial port



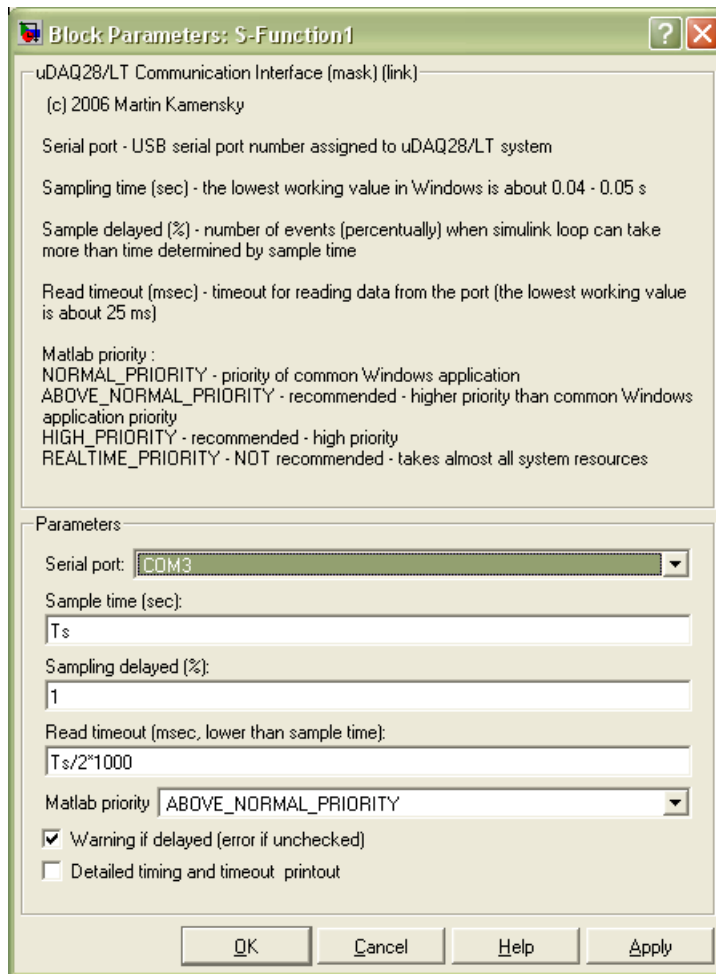


Fig. 9.8 User dialog window of communication interface

Data transfer speed: 250 kbit/s

## 9.2 Light Channel Control

The non-filtered and filtered light channels of uDAQ/28LT plant are going to be controlled in this section. Simple alternatives to linear I-controllers will be practiced.

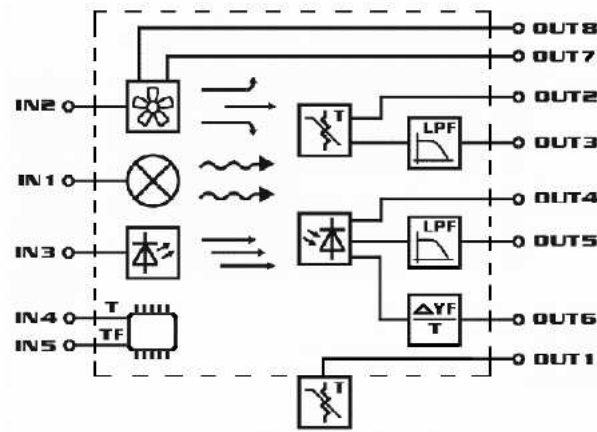


Fig. 9.9 Basic electrical diagram of thermo-optical plant uDAQ28/LT

### 9.2.1 Feedforward Control

Tasks:

- Identify non-filtered light channel parameters.
- Control non-filtered light channel using inverse process gain.
- Analyze the steady state error for various setpoint changes.
- Measure the I/O characteristics of the non-filtered light channel.

Let us start with getting to know the light channel characteristics. The non-filtered light channel represents a very fast process which can be approximated as memoryless plant. In an ideal case static feedforward control with inverse process gain should be sufficient for such process. Measure one point of the I/O characteristic to obtain the process gain.

The basic I/O Simulink model (matlabroot/udaq/udaq28LT/examples/udaq28LT\_iov2.mdl) can be used. As other alternative use the *exnum* command and choose experiment no.1 which opens up basic I/O Simulink model of the plant. Set the bulb voltage to  $u_s = 2.5$  V and run the experiment for 2 s. Put down the steady state value of the light intensity. By default it is represented by the yellow transient in the light intensity scope (Fig. 9.10).

The steady state value of the light intensity in this example is approximately  $y_s = 20$ . The process gain can be then computed as  $K = y_s/u_s$ . In this example it gives

$$K = \frac{y_s}{u_s} = \frac{20}{2.5} = 8 \tag{9.1}$$

Modify the Simulink model to use the inverse process gain to control the plant (Fig. 9.11). Do not forget to add the input saturation in the model,

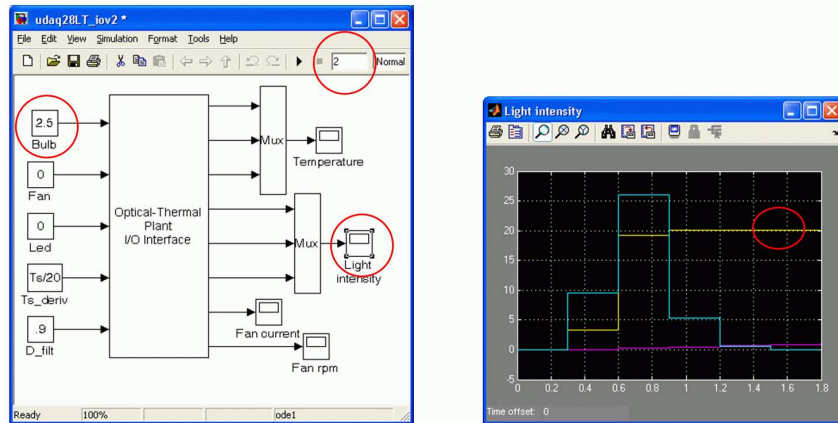


Fig. 9.10 Basic Simulink model for the udaq28/LT plant

because the bulb voltage is limited from 0 V to 5 V. Add the setpoint signal to the light intensity scope. Set the simulation time to infinity.

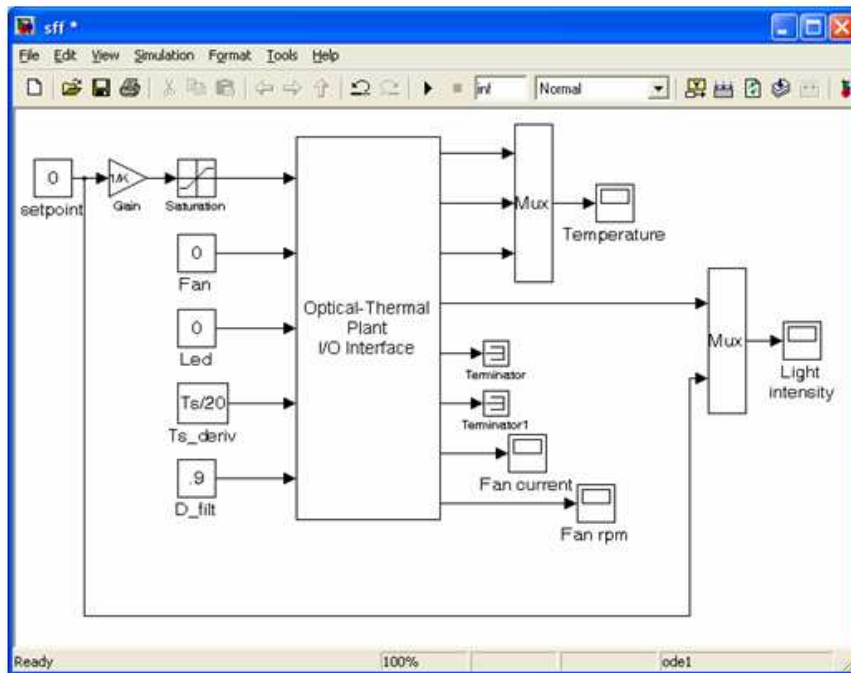
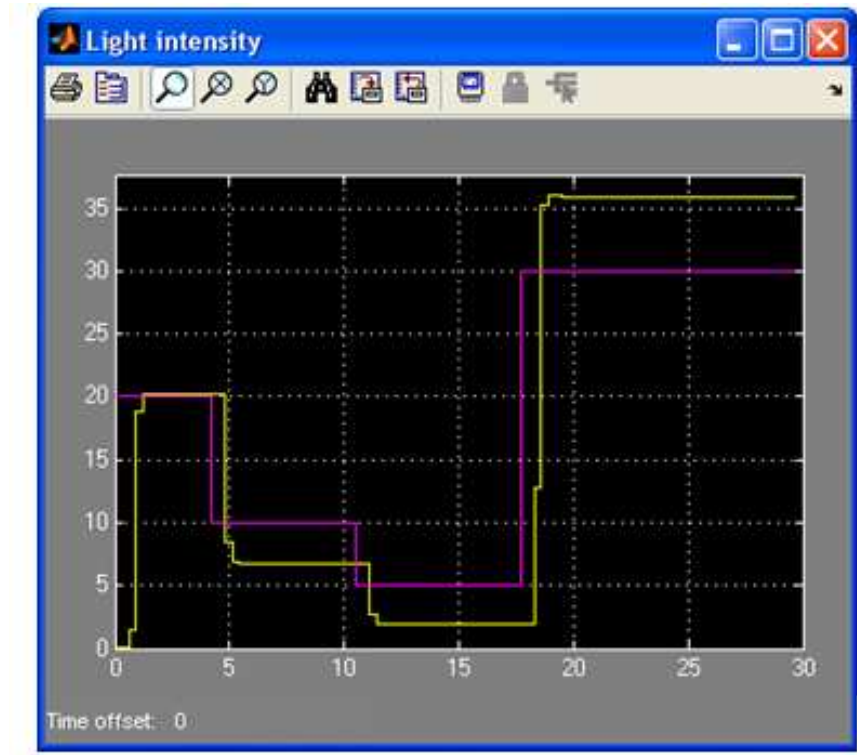


Fig. 9.11 Static feedforward control

Make multiple setpoint steps in a wide operational range. It can be done while the experiment is running.



**Fig. 9.12** Experimental results – Light intensity scope

You should observe a steady state error in several working points. The smallest steady state error can be seen around the point where the process gain was measured. It is not difficult to conclude that the process parameters vary through the operational range, in other words the I/O characteristics of the non-filtered light channel is not linear.

The `exnum` command can be used to measure I/O characteristics of the plant and to obtain the process parameters in multiple working points. Choose the experiment no.2 for I/O characteristics measurement. The following figures will give you the information on the I/O characteristic, the process gain and dead time through the operational range. Short delay in a light intensity change can be observed after a bulb voltage step. The uDAQ28/LT device converts bulb voltage steps into a steep ramp to prevent undesired disturbances in the plant. Let us approximate this delay as a dead time. After running the I/O characteristics measurement plot the step responses from

which the I/O characteristics of the light channel was obtained by the following commands:

```
stairs(yl(:,1),yl(:,2))
xlabel('t[s]')
ylabel('light_intensity')
title('Step_responses_of_non-filtered_light_channel')
```

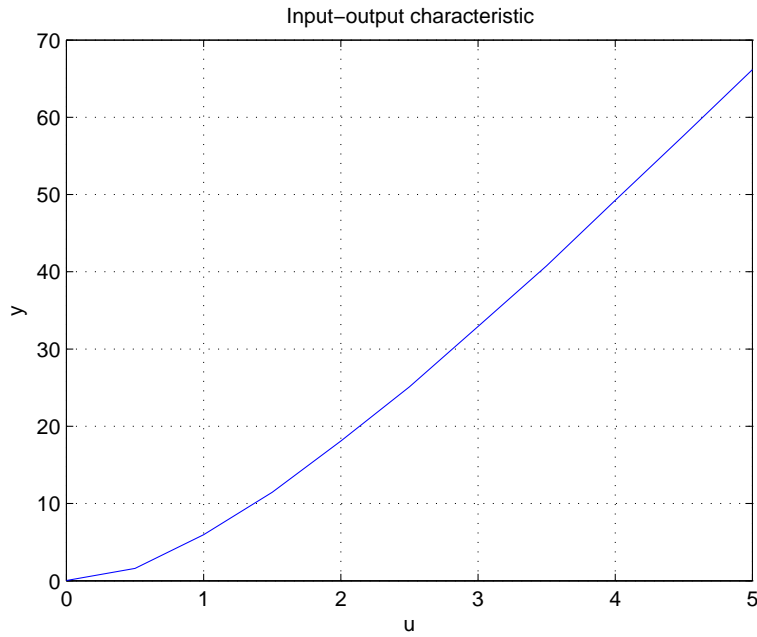


Fig. 9.13 I/O characteristics measurement results

### 9.2.2 $I_0$ Controller

Tasks:

- Add a disturbance observer to the feedforward control to compensate steady state error.
- Use the I/O characteristics measurement results to tune the controller by hand.
- Tune the controller using the performance portrait method.
- Make a step disturbance by a LED voltage step during the experiments.

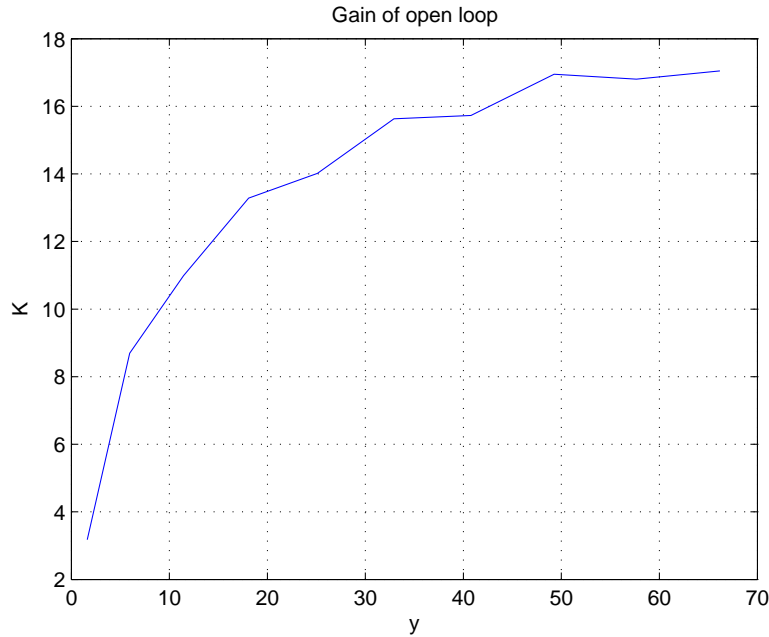


Fig. 9.14 Process gain in multiple operating points as function of the output variable

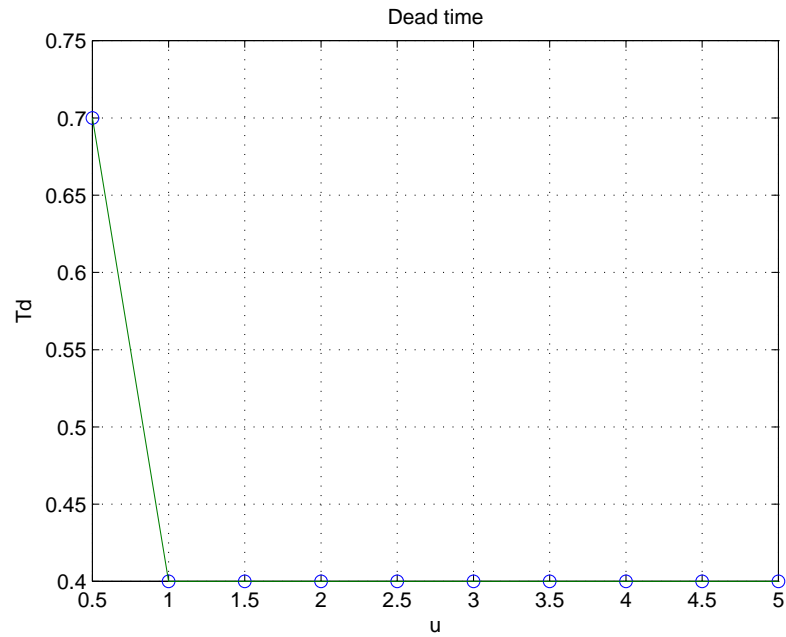
- Compare the results with various controller tuning.

The disturbance observer can be added to the static feedforward control to compensate the steady state error. To obtain a structure equivalent to I-controller, the pre-filter with time constant equal to the observer time constant has to be added as well. Let us denote the controller as  $I_0$ -controller (Fig. 9.17) and the controller with pre-filter as  $FI_0$ .

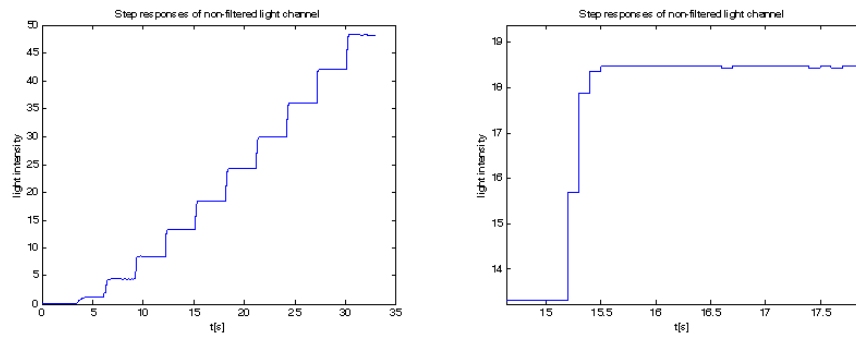
Tuning of the  $I_0$  and  $FI_0$ -controller requires information on a process gain and approximation of the non-modeled dynamics – usually by the dead time. The approximation of the delay in this example is  $T_d = 0.4$ . The filter time constant of the disturbance observer is restricted by the parasitic time delays of the non-modeled dynamics in the process. Use following formula to set up the disturbance observer filter time constant:

$$T_{fil} = eT_d \tag{9.2}$$

Try to use multiple process gains for controller tuning, to improve control quality. Use the lowest, the average and the maximum process gain from Fig. 9.13. Choose experiment no.3 with the `exnum` command to control non-filtered optical channel by  $FI_0$ -controller. You will be prompted for the process gain value and the delay time constant. In Fig. 9.18 there is the Simulink model which should pop up when the correct experiment starts. The exper-



**Fig. 9.15** Dead time in multiple operating points as function of the input variable



**Fig. 9.16** Step responses of the non-filtered optical channel – overview and a detail of one step response. These are the responses to 0.5V bulb voltage steps made in 3s intervals.

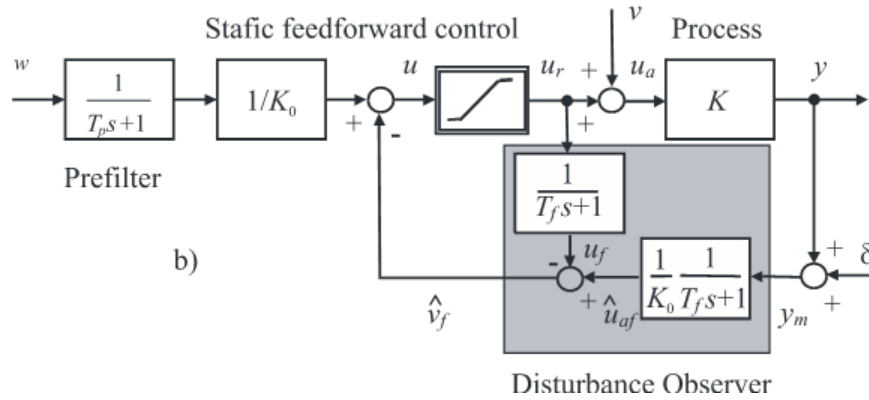


Fig. 9.17 FI<sub>0</sub>-controller, structure equivalent for  $T_p = T_f$  to I-controller

iment will run with multiple setpoint steps. Feel free to modify the model to make your own setpoint steps sequence. The goal is to achieve quick non-overshooting transients.

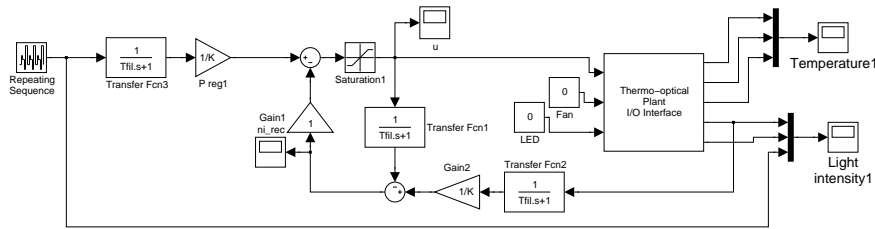


Fig. 9.18 FI<sub>0</sub>-controller – Simulink model

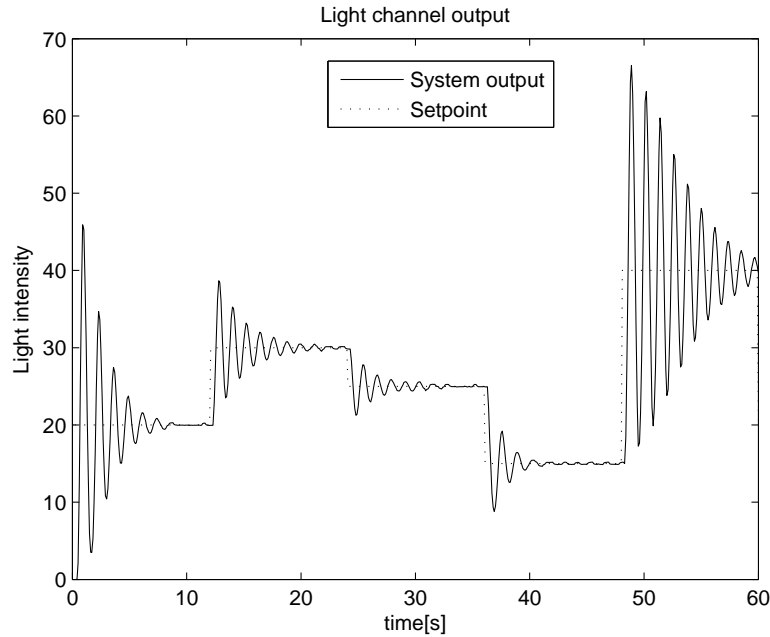
The experimental results can be plot using following commands.

```
figure
stairs(yl(:,1),yl(:,2), 'k')
hold on
stairs(yl(:,1),yl(:,4), 'k:')
xlabel('t[s]')
ylabel('light_intensity')
legend('system_output', 'setpoint')
```

Using the recommended three process gains in the experiment you should obtain similar results to Fig. 9.19, 9.20 Fig. 9.21,9.22, and Fig. 9.23,9.24.

For the plant used in this example the desired control quality was achieved when the process gain  $K = 17$  was used. The control results can be seen in Fig. 9.23.





**Fig. 9.19** Experimental results for  $K=3$

From the experience with the previous experiments one can assume that a higher value of the process gain leads to lower overshoot with slower transients and vice versa. After getting some experience with the controller, let us practice robust controller tuning. At first put down the intervals in which the process gain and the dead time range. The data from Fig. 9.14 and Fig. 9.15 can be used. In this example the dead time  $T_d$  ranges in interval  $[0.4, 0.7]$  and the process gain ranges in interval  $[3, 17]$ . Now determine the interval in which normalized variable  $\kappa$  ranges.  $\kappa$  represents normalized variable

$$\kappa = K_0/K \quad (9.3)$$

where  $K_0$  stands for the process gain used in the controller,  $K$  corresponds to real process gain, which varies through the operational range. The goal is to fit the uncertainty box into area of performance portrait (Fig. 9.25) where the controller gives monotonic transients. Try to use two limit values of process gain for  $K_0$ , in this example it gives

$$K_0 = K_{min} = 3 \quad (9.4)$$

which leads to  $\kappa = [3/17, 1]$ .

For the maximal value of  $K_0$

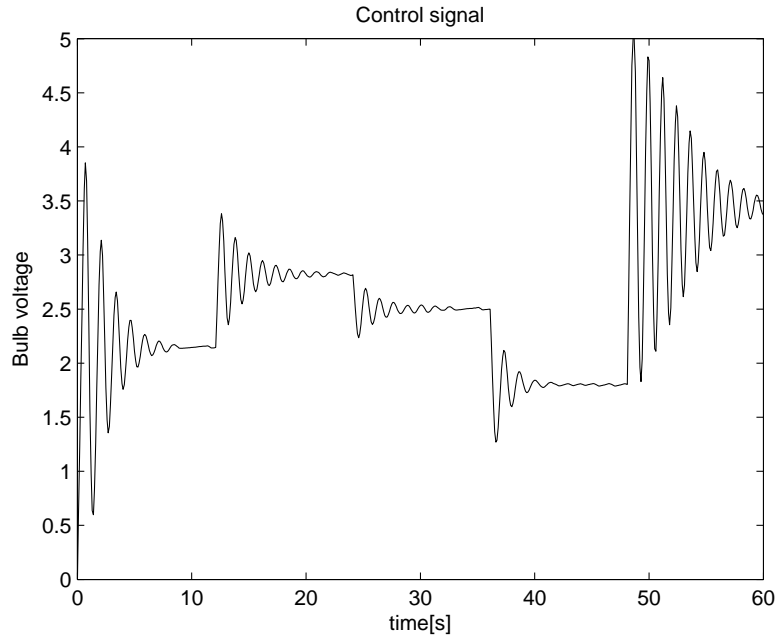


Fig. 9.20 Experimental results for  $K=3$

$$K_0 = K_{max} = 17 \tag{9.5}$$

it gives  $\kappa = [1, 17/3]$ .

$$\Omega = T_d/T_f \tag{9.6}$$

Calculate filter time constant  $T_f$  for both selections of  $K_0$ . For  $K_0$  (9.4) it yields

$$\Omega = 0.0649 \tag{9.7}$$

$$T_f = T_d/\Omega = 0.7/0.0649 = 10.7825 \tag{9.8}$$

For  $K_0$  (9.5) it yields

$$\Omega = 0.3679 \tag{9.9}$$

$$T_f = T_d/\Omega = 0.7/0.3679 = 0.7e = 1.9028 \tag{9.10}$$

Verify controller tuning by real experiment for various setpoint steps. Add a disturbance using a LED to verify input disturbance compensation. You should observe similar transients using both controller tunings (Fig. 9.26). The 2 V LED voltage step was made at time 105 s. Following table summarizes the robust  $FI_0$ -controller tuning for various overshooting tolerances.

Questions:

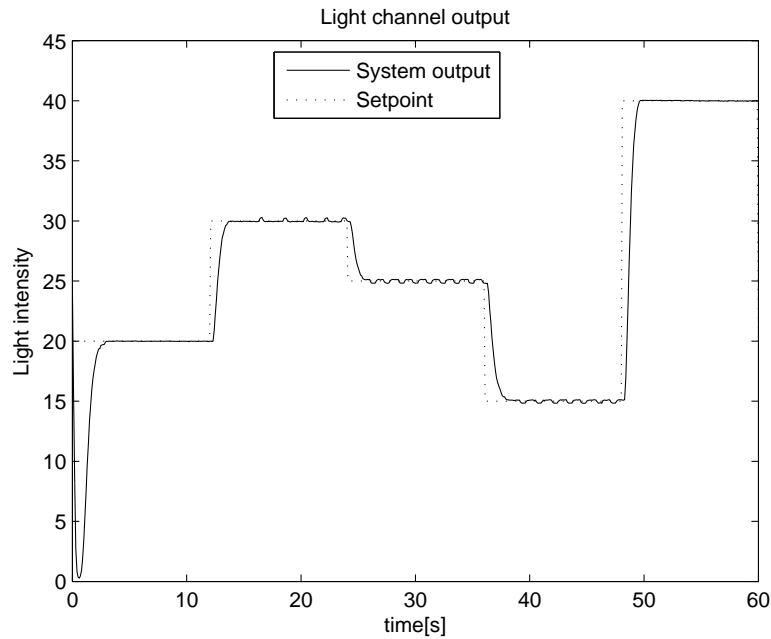


Fig. 9.21 Experimental results for  $K=10$

Table 9.1 Controller tuning

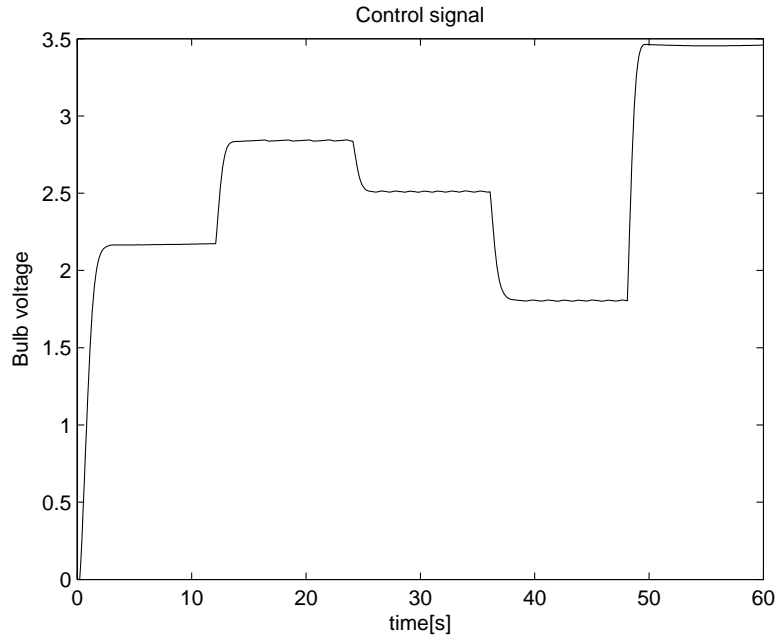
$\epsilon_y$ [%]	10	5	4	2	1	0.1	0.01	0.001	0	0
$\tau = \kappa/\Omega$	1.724	1.951	2.0	2.162	2.268	2.481	2.571	2.625	2.703	2.718
$q = \Omega/\kappa$	0.58	0.515	0.5	0.465	0.441	0.403	0.389	0.381	0.37	0.368

- What type of plant is  $FI_0$ -controller suitable to control for?
- Which plant's parameter restricts the controller's dynamics?
- Was there any overshooting in the experiments?
- Why the performance of different controller tunings in Fig. 9.26 is almost the same?

### 9.2.3 Filtered Predictive $I_0$ Controller

Tasks:

- Add a dead time to the non-filtered light channel output.
- Modify the  $I_0$ -controller to compensate the delay.
- Tune the controller using a performance portrait.

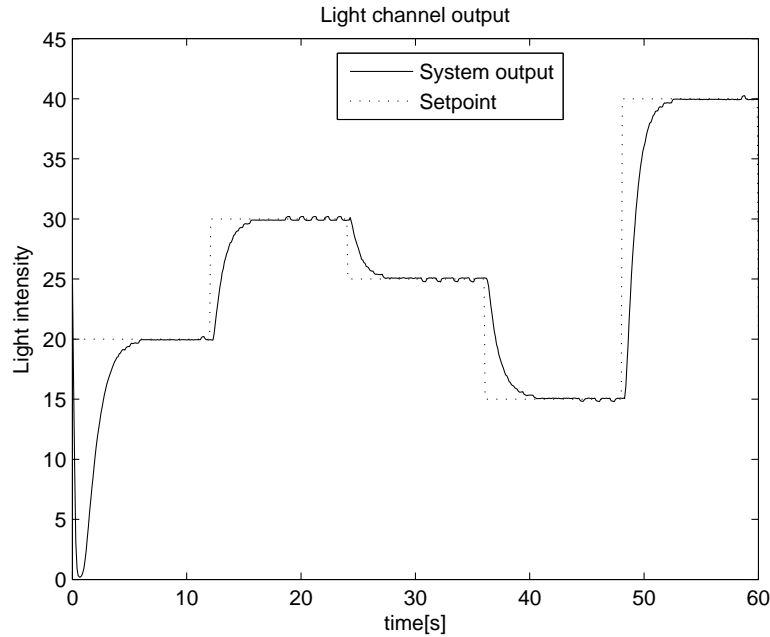


**Fig. 9.22** Experimental results for  $K=10$

- Make a disturbance by a LED voltage step during the experiments.
- Compare the control quality with the  $FI_0$ -controller using a real experiment.

Tuning of the closed loop systems involving dead-time still represents a challenging domain of control research. an increase of the dead-time values with respect to the dominant plant time constant leads in the loops with PID controllers to rapid performance deterioration. Therefore filtered predictive  $I_0$  (FPr $I_0$ )-controller will be used in this exercise. Under the FPr $I_0$ -controller we will understand the static feedforward control with the gain  $1/K_0$  extended by the input disturbance reconstruction and compensation (Fig. 9.27) with the disturbance observer filter time constant  $T_f$  and by the pre-filter with the time constant  $T_p = T_f$ .

Robust tuning of the FPr $I_0$ -controller may again be done by the performance portrait. The information on plant parameters is needed. For the plant used in this exercises the performance portrait for undelayed plant output is in Fig. 9.29, the delayed plant output performance was analyzed in Fig. 9.30. Compare the performance of filtered predictive  $I_0$ -controller vs  $FI_0$ -controller. Use additional transport delay when controlling a non-filtered optical channel. In the following example 5 s transport delay was added to the non-filtered light channel output, so the transport delay of the plant ranges from 5.4 to



**Fig. 9.23** Experimental results for  $K=17$

5.7 s. The process gain still range in interval  $[3,17]$ . For 1% overshooting tolerance it yields to filtered predictive  $I_0$ -controller parameters:

$$T_{d0} = 6.48 \quad (9.11)$$

$$T_f = 4.28 \quad (9.12)$$

$$K_0 = 14.15 \quad (9.13)$$

For  $FI_0$ -controller it gives

$$T_{d0} = 5.7 \quad (9.14)$$

$$T_f = eT_{d0} \approx 16 \quad (9.15)$$

$$K_0 = 17 \quad (9.16)$$

The performance of both controllers is compared in Fig. 9.31, 9.32. Feel free to make this experiment with larger dead time e.g. 10 s.

Questions:

- What type of plant is  $FPrI_0$ -controller suitable to control for?
- Was there any overshooting in output and input transients during the experiments?
- Which controller performed better in disturbance rejection?

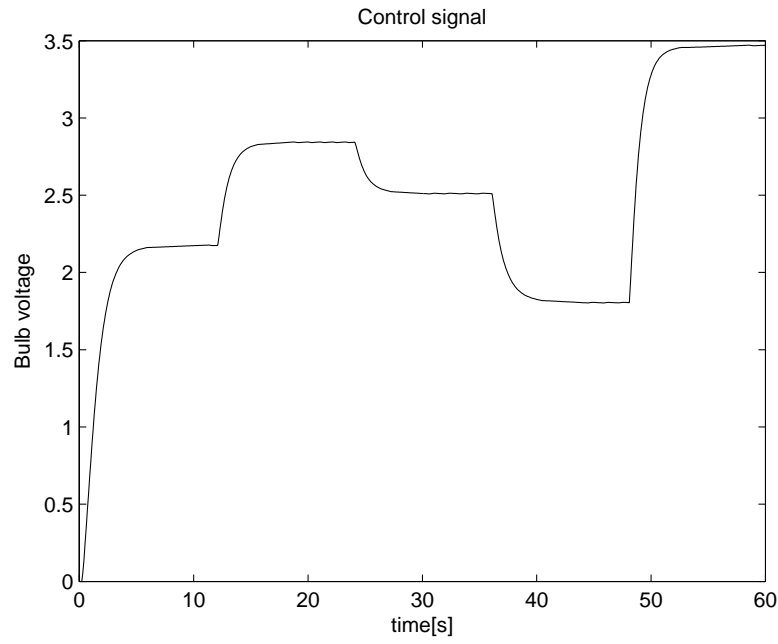


Fig. 9.24 Experimental results for  $K=17$

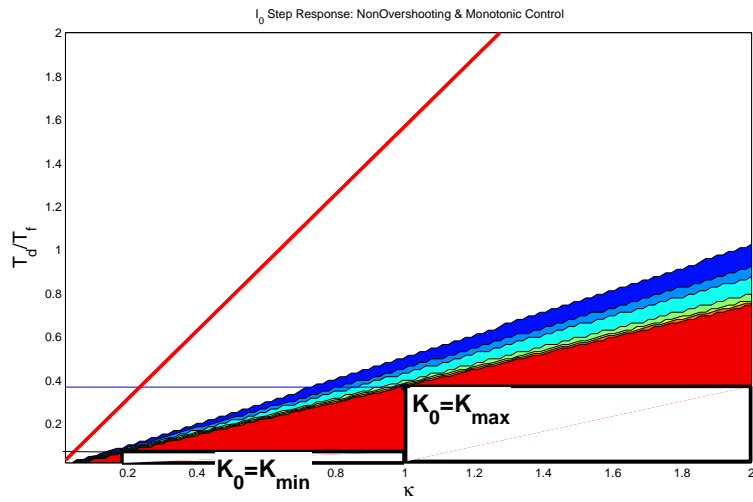


Fig. 9.25 Performance portrait for  $FI_0$ -controller

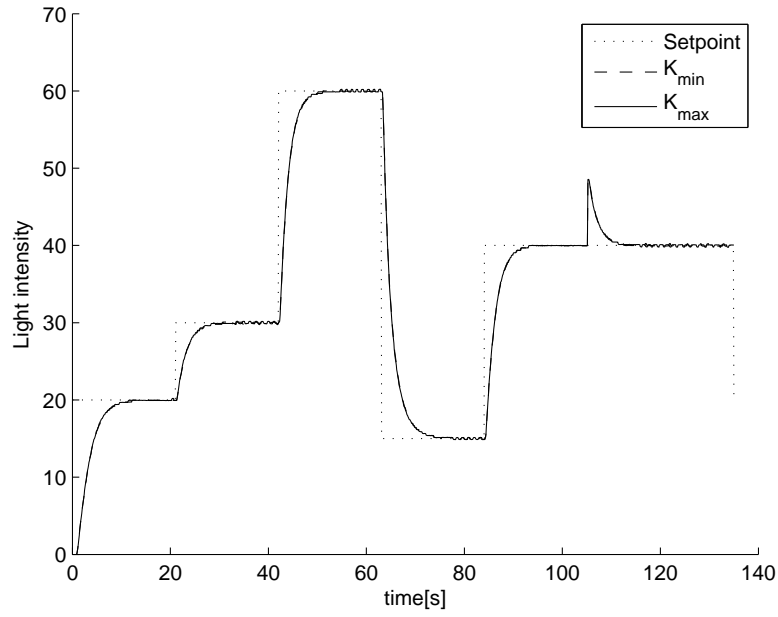


Fig. 9.26 Real experimental results for FI<sub>0</sub>-controller

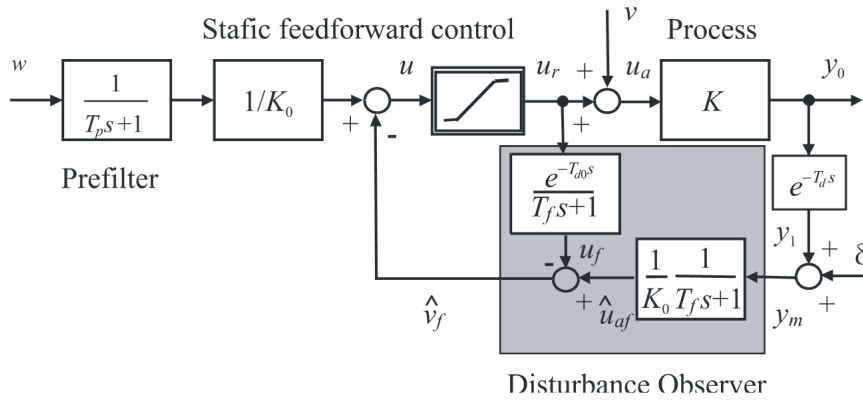


Fig. 9.27 FPrI<sub>0</sub>-controller

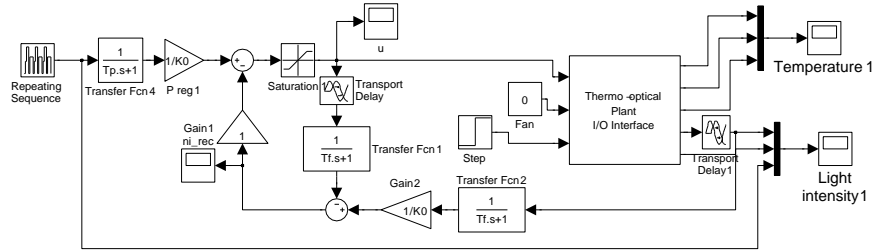


Fig. 9.28 FPrI<sub>0</sub>-controller - Simulink model

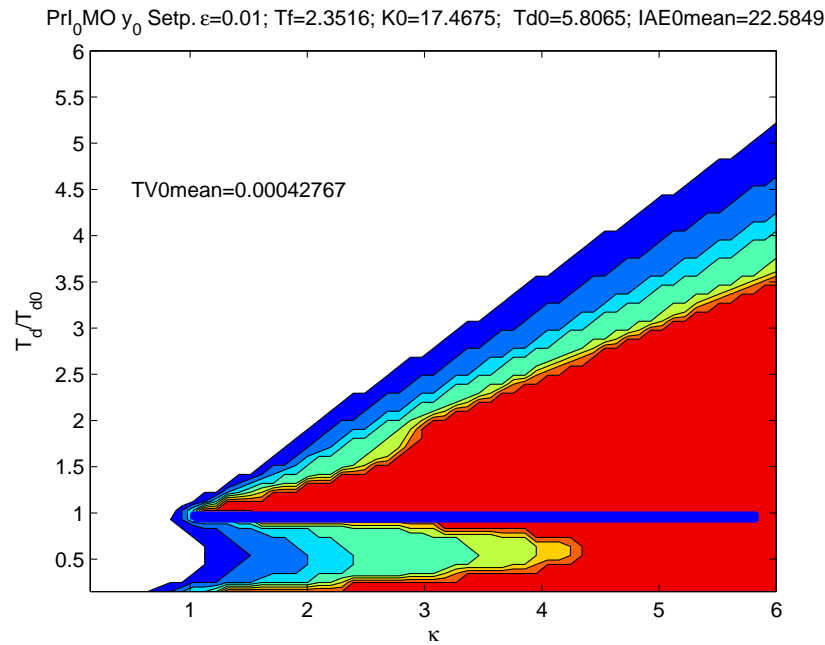


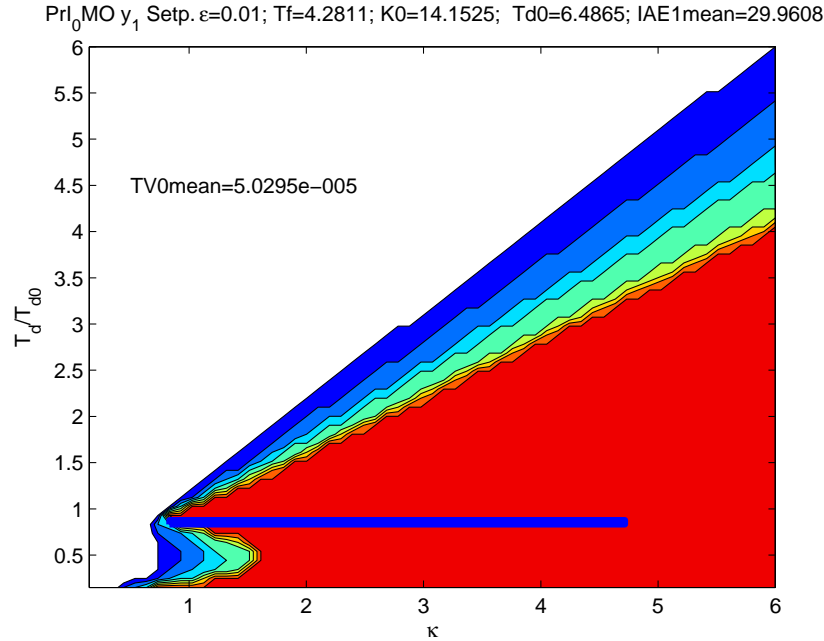
Fig. 9.29 Performance portrait – FPrI<sub>0</sub>-controller

### 9.2.4 PI<sub>0</sub> and FPI<sub>0</sub> Controllers

Tasks:

- Identify the parameters of filtered light channel.
- Use PI<sub>0</sub>-controller to compensate the delay of the filtered light channel.
- Tune the controller using a performance portrait.
- Make a disturbance by a LED voltage step during the experiments with the filtered light channel.





**Fig. 9.30** Performance portrait – FPrI<sub>0</sub>-controller

- Compare the control quality with the I<sub>0</sub>-controller using a real time experiment.

In practice it is often not efficient and sufficient to compensate large time constant's influence just by restricting the closed loop bandwidth. Active compensation of dominant loop time constant leads to control structures such as PI<sub>0</sub>-controller (Fig. 9.33, 9.34).

The output of filtered optical channel will be used to practice FPI<sub>0</sub>-controller tuning. An analogue first order filter is used for non-filtered light channel filtering. The process can so be approximated as

$$G(s) = \frac{K}{T_1 s + 1} e^{-T_d s} \quad (9.17)$$

The time constant  $T_1$  represents a analogue filter time constant, dead time  $T_d$  is used to approximate the lag between a bulb voltage step and the corresponding change in the light intensity. Obtain the parameters of filtered light channel. The exnum command with experiment no.5 can be used.

For the plant used in this example it gives

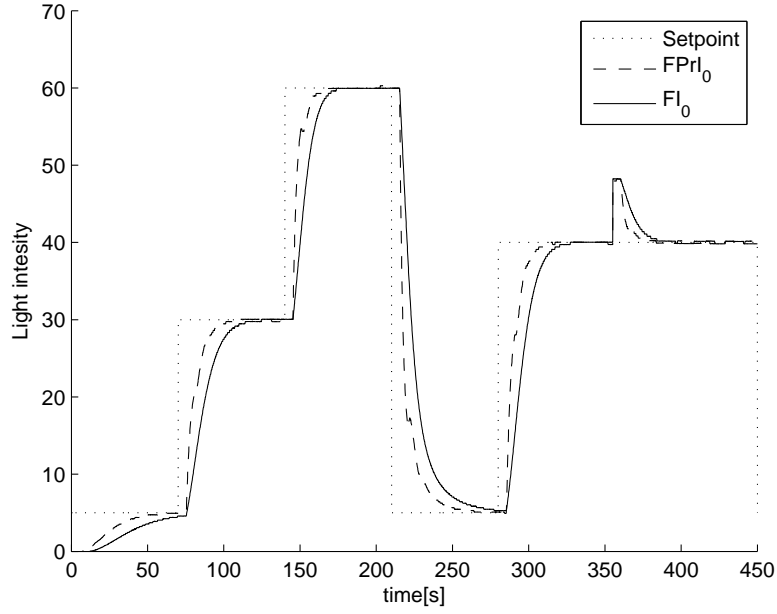


Fig. 9.31 FPrI<sub>0</sub> – controller vs FI<sub>0</sub>-controller under 5s transport delay, plant output

$$\begin{aligned}
 K &\in [8.8728, 19.8939] \\
 T_1 &\in [17.0096, 25.8687] \\
 T_d &\in [0, 0.6]
 \end{aligned}$$

The dead time  $T_d$  is relatively small comparing to the process time constant  $T_1$  (we have so-called lag dominant plant), thus it can be neglected in further calculations. Figs. 9.35 and 9.36 show how the process gain and time constant vary through the operational range independence on the plant input.

Again the controller tuning can be done using the performance portrait method (Fig. 9.37). It is best to choose

$$\begin{aligned}
 T_{10} &= \max(T_1) \\
 K_{10} &= \max(K)
 \end{aligned}
 \tag{9.18}$$

which for this plants is

$$\begin{aligned}
 T_{10} &= 25.8687 \\
 K_0 &= 19.8939
 \end{aligned}
 \tag{9.19}$$

Try multiple disturbance observer filter time constants  $T_f$ . In this example following  $T_f/T_{10}$  ratios were used:

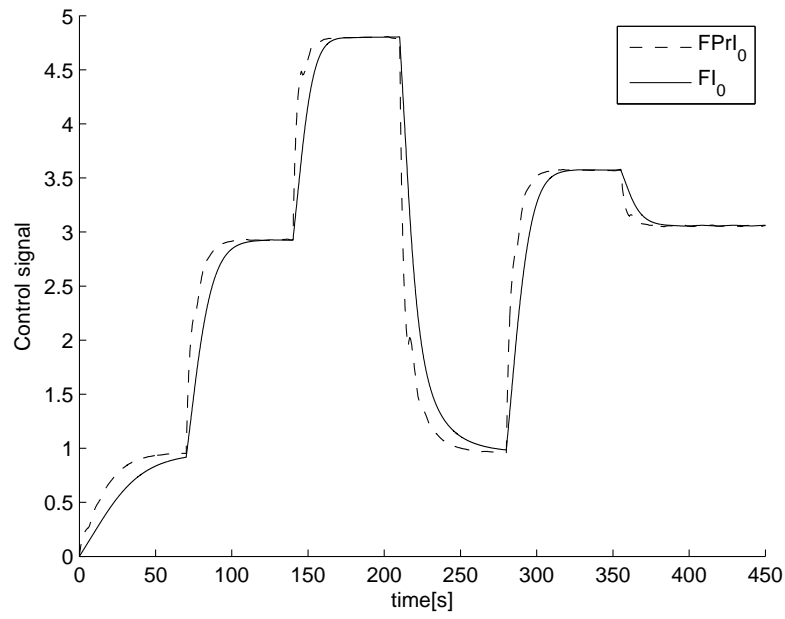


Fig. 9.32 FPI<sub>0</sub>-controller vs FI<sub>0</sub>-controller under 5s transport delay, control signal

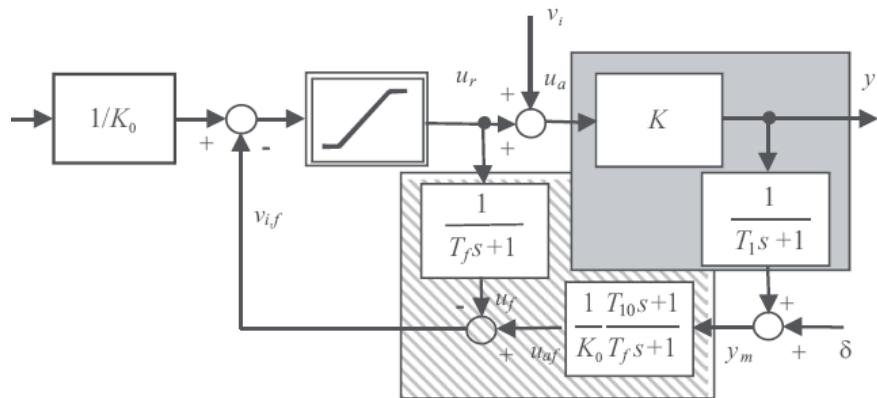


Fig. 9.33 FPI<sub>0</sub> controller

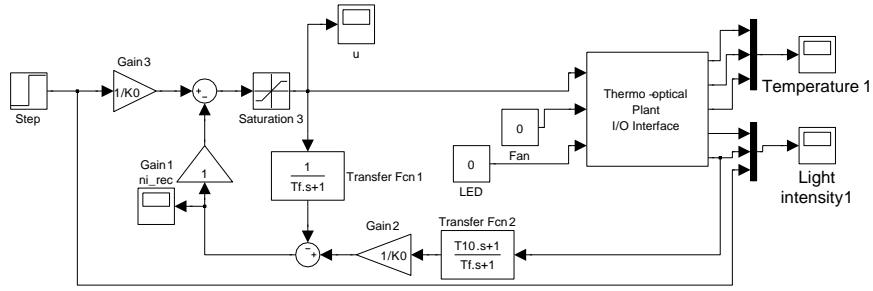


Fig. 9.34 PI<sub>0</sub>-controller: Simulink Model; to get FPI<sub>0</sub> controller you need to add pre-filter to the reference setpoint input

Gains corresponding to approximation of step responses produced by incremental input chang

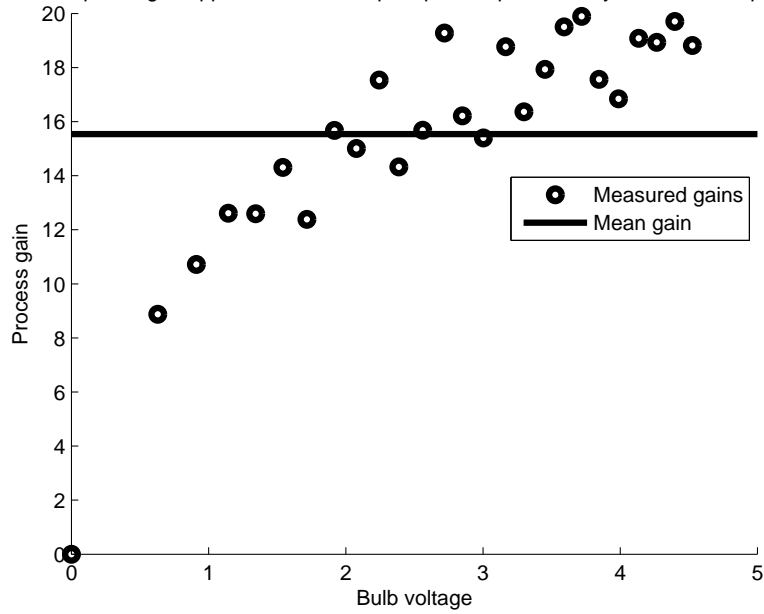
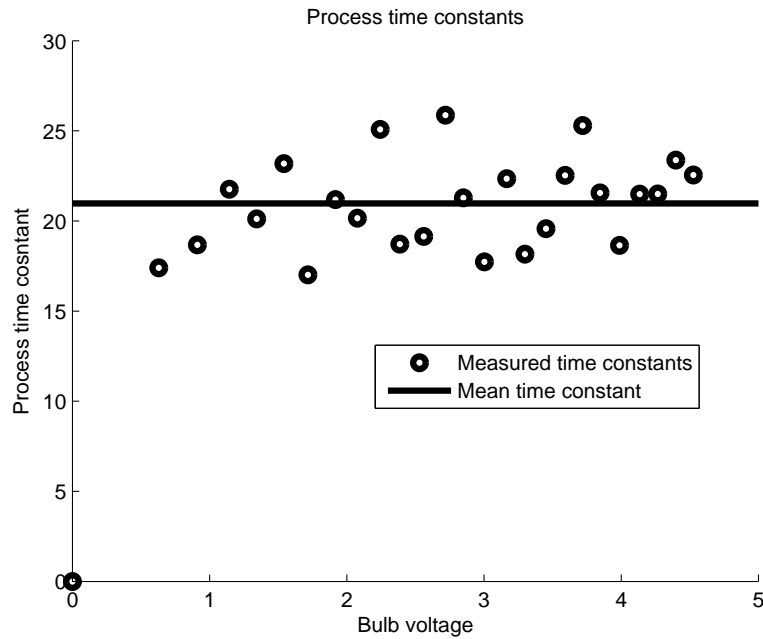


Fig. 9.35 Process gain in several operation points as function of the plant input

$$T_f/T_{10} = \{0.8, 0.6, 0.06\} \tag{9.20}$$

$T_f/T_{10} = 0.8$  and  $T_f/T_{10} = 0.6$  should give transients of the undelayed system output with up to 2% overshooting and the delayed system output should not overshoot.  $T_f/T_{10} = 0.06$  corresponds to controller tuning where

$$T_f = e^1 \cdot \max(T_d) = 1.6310 \tag{9.21}$$



**Fig. 9.36** Process time constant in several operation points as function of the plant input

This tuning can yield to transients with approximately 10% overshooting of the undelayed system output and the delayed system output should not overshoot (see the performance portrait in Fig. 9.37).

Compare the results with the  $FI_0$ -controller (Fig. 9.17). Use the following  $FI_0$ -controller tuning:

$$\begin{aligned} K_0 &= \max(K) \\ T_f &= e^1 \cdot \max(T_1) \end{aligned} \quad (9.22)$$

Make several setpoint steps, make a LED voltage step as well when the system output is settled. Do not forget to keep more time between setpoint steps when using  $I_0$ -controller. It is useful to make a simulation first to determine setpoint step time interval sufficient to settle the system output between them. The experiments results are shown in Figs. 9.38, 9.39, 9.40. Compare the overshooting for setpoint step and disturbance step. The detailed view on these transients is shown in Figs. 9.41, 9.42, 9.43, and 9.44.

Questions:

- What type of plant are  $PI_0$  and  $FPI_0$ -controllers suitable to control for?
- Analyze the amount of overshooting in the experiments.
- Which controller performed better in disturbance rejection?

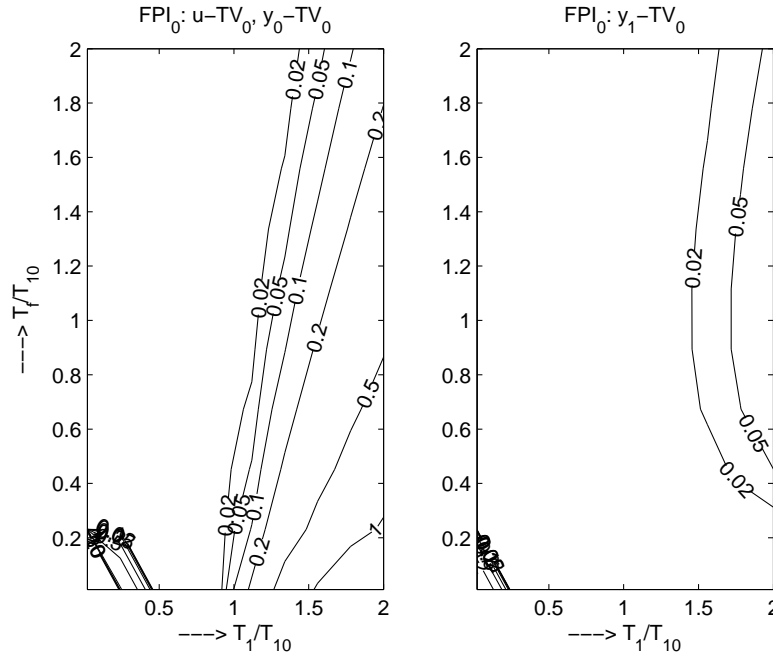


Fig. 9.37 Performance portrait of the FPI<sub>0</sub>-controller.

- Was there any advantage of using FI<sub>0</sub> over FPI<sub>0</sub> – e.g. in noise sensitivity?

### 9.2.5 PI<sub>1</sub> controller

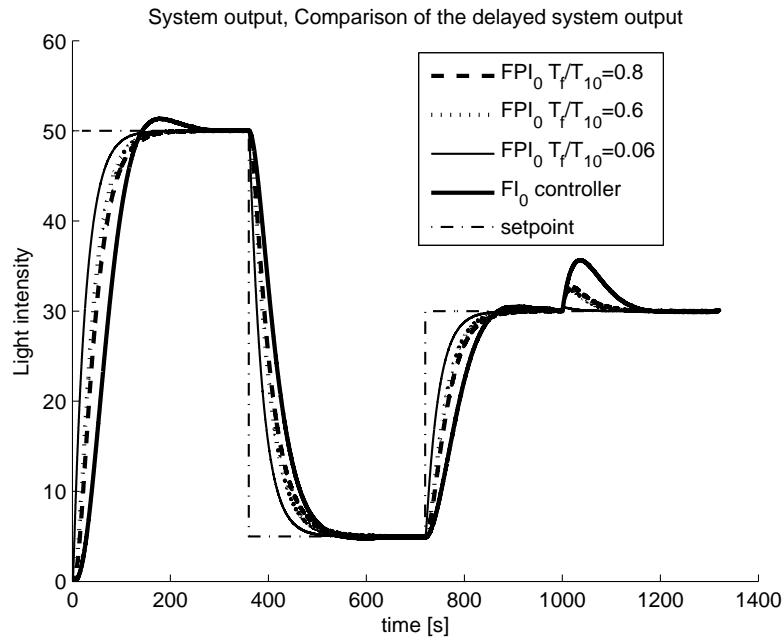
Tasks:

- Control the filtered light channel output by PI<sub>1</sub>-controller.
- Analyze the control quality for various controller tunings.
- Compare results with control performance of previous controllers.

The PI<sub>1</sub>-controller structure and Simulink model are in Figs. 9.45, 9.46. PI<sub>1</sub> may also be used for controlling unstable plant. To cover all stable and unstable plants by one transfer function, it is necessary to use the pole-zero form instead of the time-constant one and to express the plant as

$$G(s) = \frac{K_s}{s + a} e^{-T_d s} \tag{9.23}$$

For robust controller tuning, the performance portrait method could again be used. For simple nominal tuning use the following rules based on the notion of the so called equivalent poles  $\alpha_e$  of the proportional controller and  $\alpha_{e,I}$  of



**Fig. 9.38** Experimental results – delayed system output

the controller with disturbance compensation (I-action):

$$\alpha_e = -(1 + aT_d)^2 / (4T_d) \quad (9.24)$$

$$\alpha_{eI} = \alpha_e / 1.3 \quad (9.25)$$

$$P = -(\alpha_{eI} + a) / K_s \quad (9.26)$$

$$T_f = e^1 T_d \quad (9.27)$$

Plant parameters from FOPTD approximation (9.18) can be used, whereby  $K_s = K/T$  and  $a = 1/T$ . The experimental results for the plant used in this chapter are in Figs. 9.47, 9.48, 9.49, 9.49.

Questions:

- Which process gain used for controller tuning gave better control quality?
- What was the most obvious difference in control quality compared to DC0 controllers?
- What was the most obvious difference in control signal shape compared to DC0 controllers?
- Was there any overshooting in the experiments?

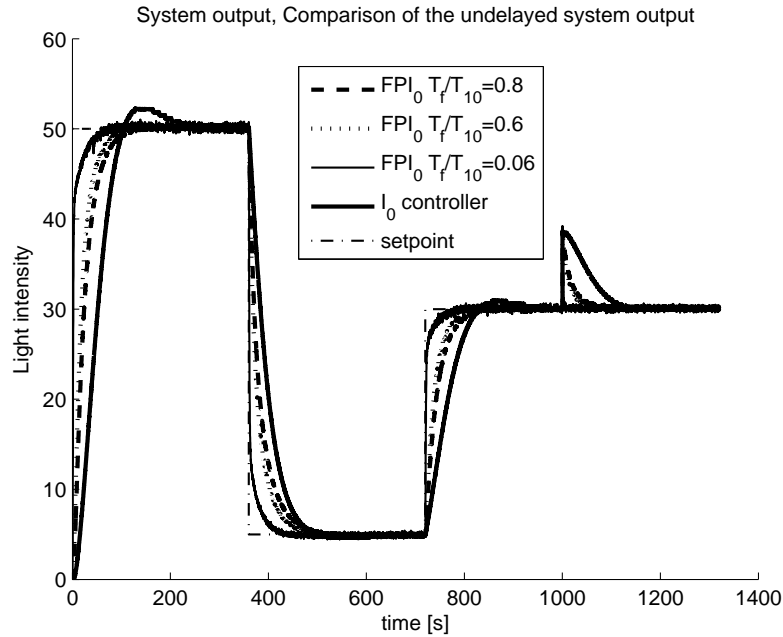


Fig. 9.39 Experimental results – undelayed system output

### 9.2.6 Filtered Smith Predictor (FSP)

Tasks:

- Control the filtered light channel output by FSP.
- Analyze the control quality for various controller tunings.
- Compare results with control performance of  $PI_1$ -controller .

The FSP was originally proposed in Normey-Rico et al. (1997) for stable FOPDT processes to improve robustness of the traditional SP. Later, the disturbance filter  $F_r(s)$  has been also proposed to decouple the reference setpoint and the disturbance response and to stabilize the controller loop in case of unstable and integral plants Normey-Rico and Camacho (2009). It may be interpreted as a structure with the dynamical feedforward control and the reference plant model Åström and Hägglund (2005); Visioli (2006), or the 2DOF IMC structure. The unified approach to designing FSPs for the FOPDT plants introduced in Normey-Rico et al (2009); Normey-Rico and Camacho (2009) considers compensation of an output disturbance by correction of the reference value, whereby the disturbance is reconstructed by using the PPM. However, despite to the proclaimed unification, it separately presents solutions corresponding to stable, integral and unstable plants.

Use FSP structure from Fig. 9.51 with filter



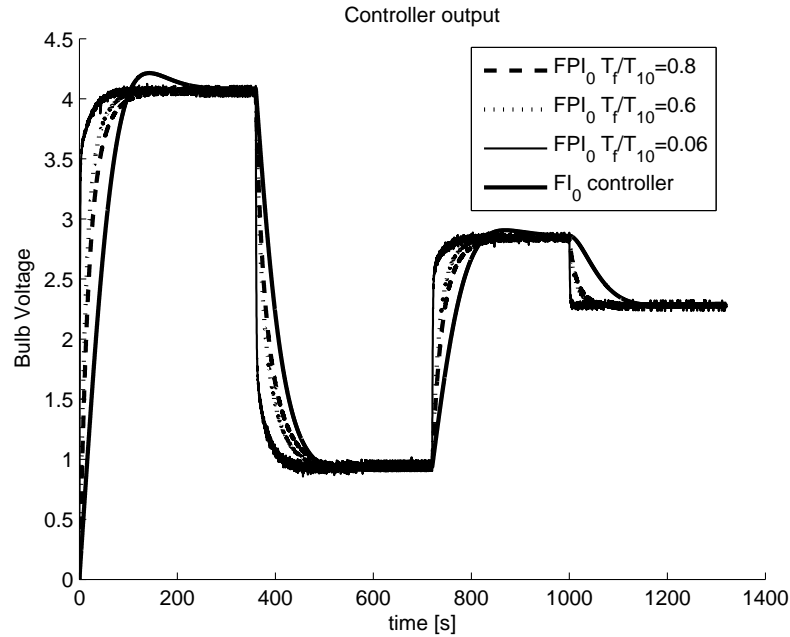


Fig. 9.40 Experimental results – control signal

$$F_{r1}(s) = \frac{1 + \beta_{11}s}{1 + T_f s} \quad (9.28)$$

Use FOPDT approximation (9.18), whereby  $K_s = K/T$  and  $a = 1/T$ . Choose  $K_{s0} = \max(K_s)$ ,  $\theta = \max(T_d)$ ,  $T_{10} = \max T_1, a_0 = \min(a)$ . Set up the experiments in the same way as in the previous exercise to be able to compare the results. For controller tuning use following rules.

The P-action should be set to

$$K_p = (1/T_r - a_0)/K_{s0} \quad (9.29)$$

Filter parameter  $\beta_{11}$  set to

$$\beta_{11} = T_{10} \left( 1 - (1 - T_f/T_{10})(1 - T_r/T_{10})e^{-\theta/T_{10}} \right) \quad (9.30)$$

Try multiple  $T_r$  and  $T_f$  settings e.g.  $T_r = T_{10}/\{2, 4, 8, 16\}$ ,  $T_f = T_r/\{2, 4, 8\}$ . In this example various  $T_r$  settings were used. The filter time constant was set to  $T_f = T_r/4$ . Experimental results are shown in Figs. 9.52, 9.53.

Questions:

- Was there any overshooting in the experiments?
- How did the increasing of parameter  $T_r$  affect control quality?

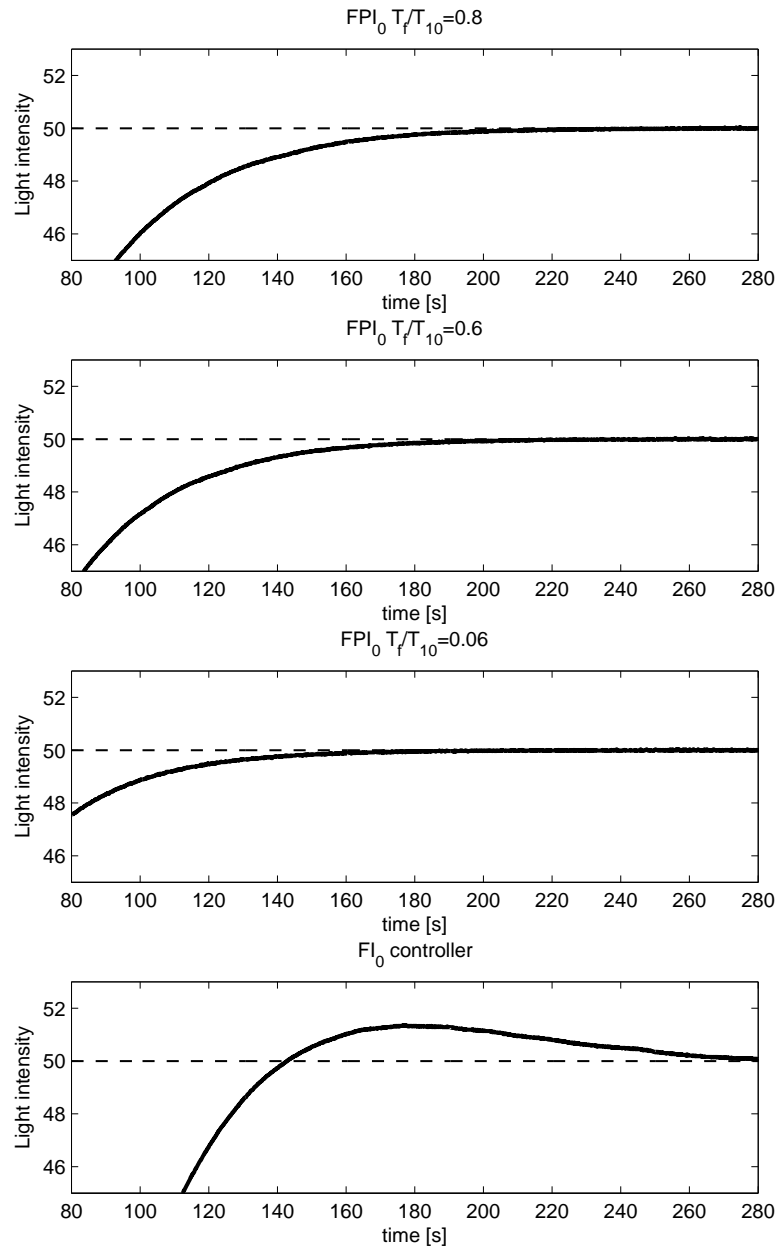


Fig. 9.41 Experimental results detail – delayed system output, setpoint step

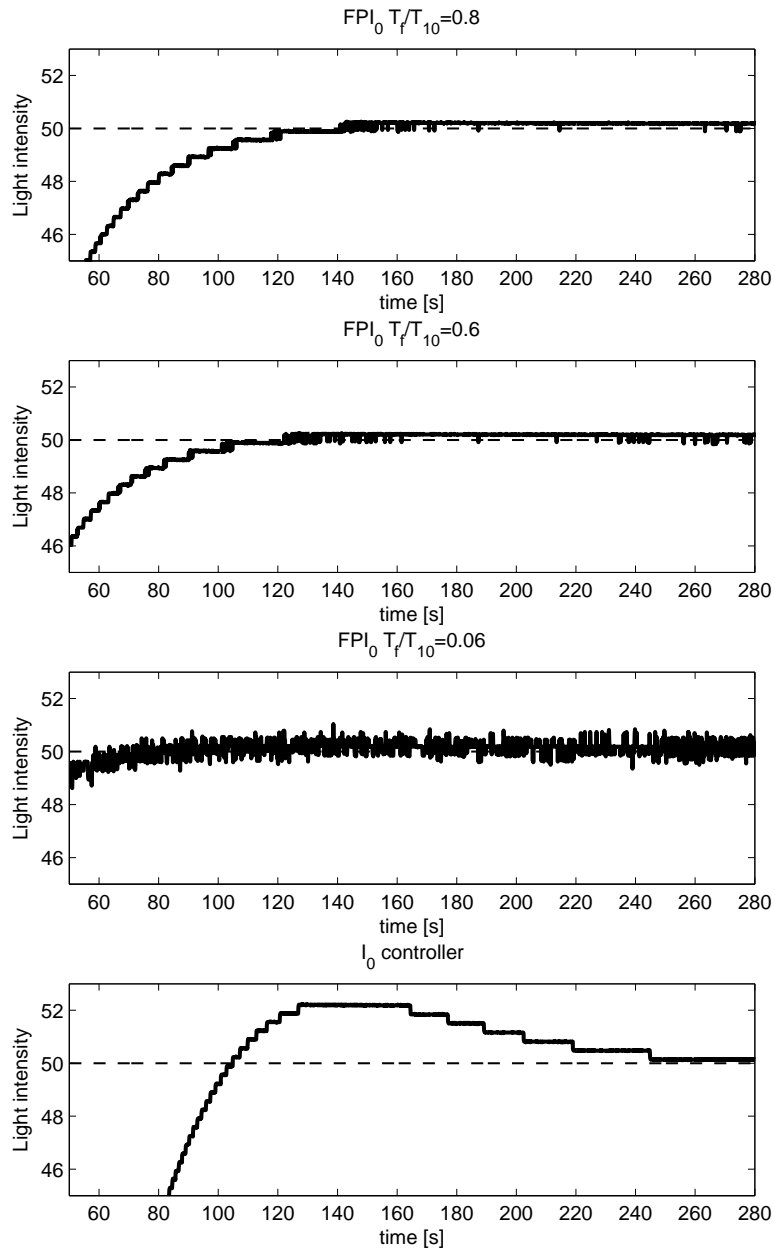


Fig. 9.42 Experimental results detail – undelayed system output, setpoint step

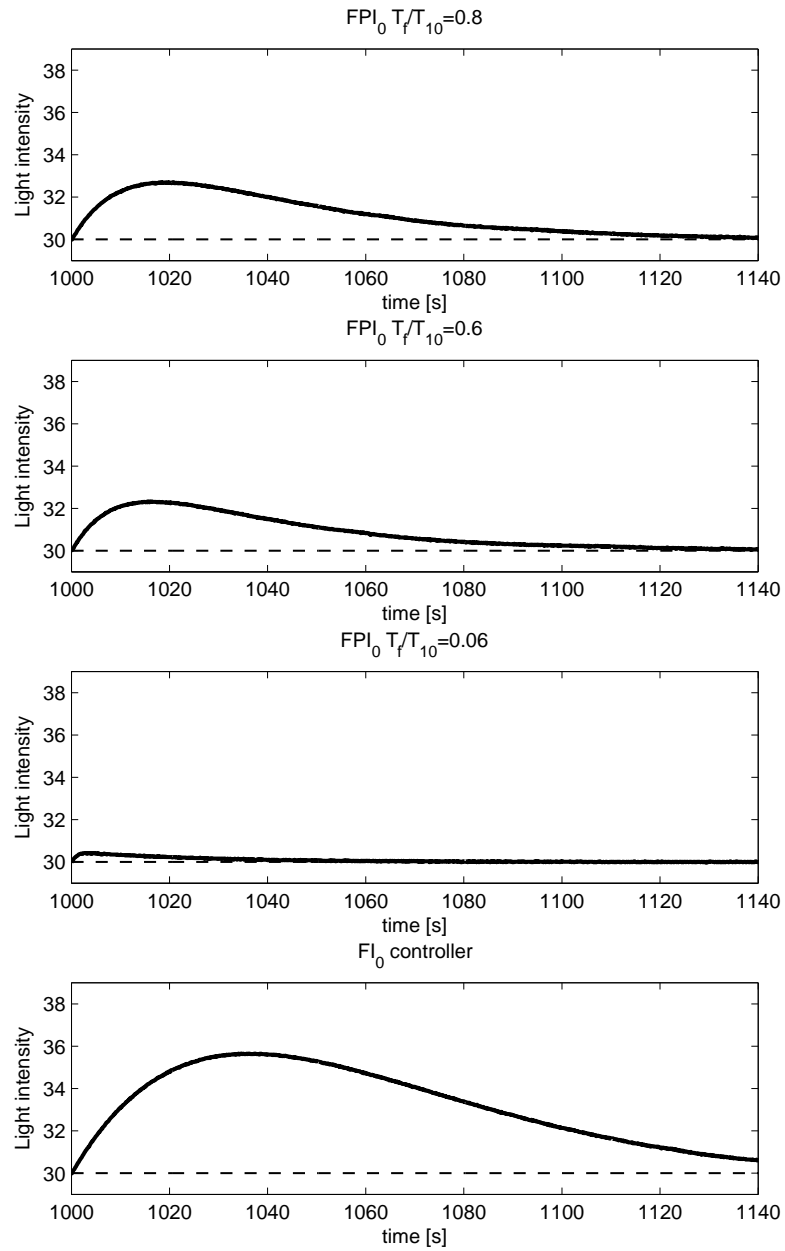


Fig. 9.43 Experimental results detail – delayed system output, disturbance step

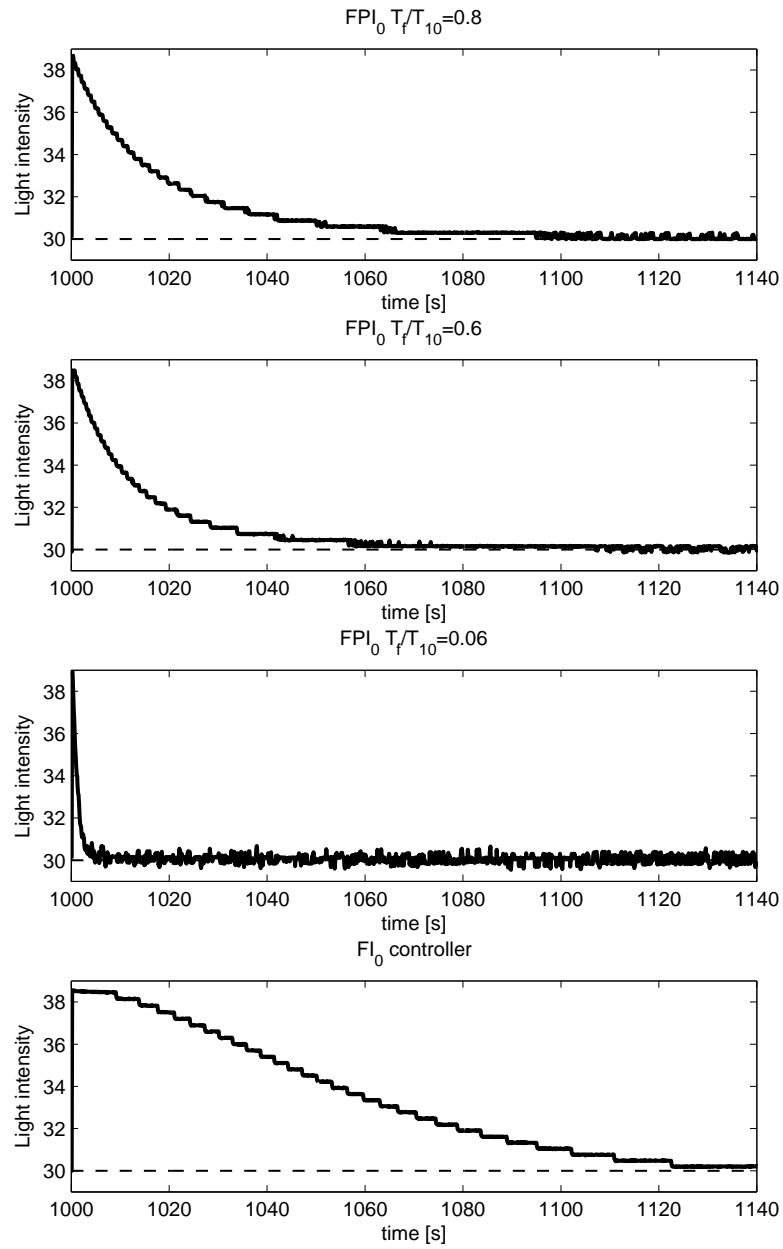


Fig. 9.44 Experimental results detail – undelayed system output, disturbance step

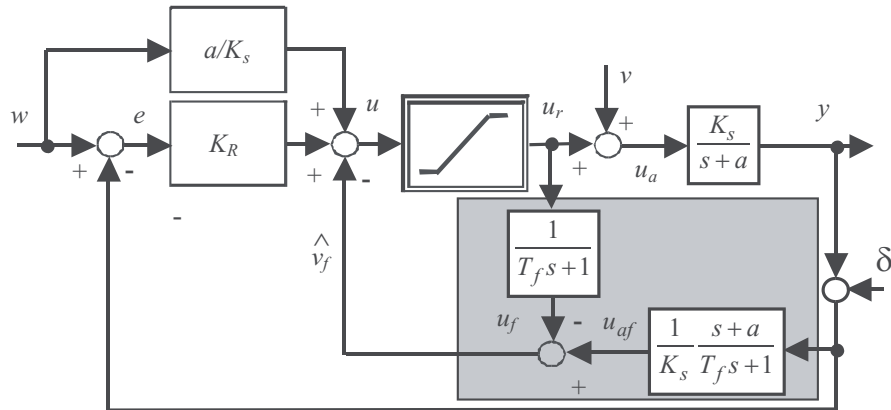


Fig. 9.45 PI<sub>1</sub>-controller

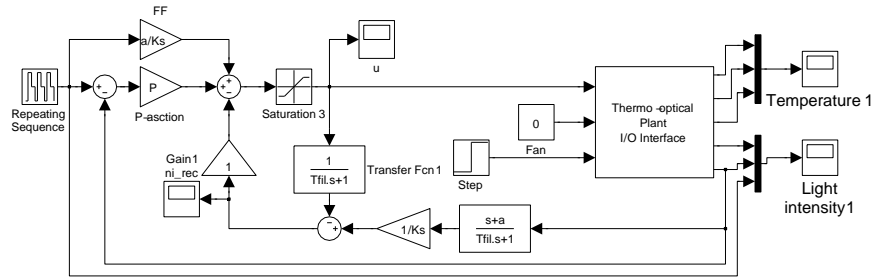


Fig. 9.46 PI<sub>1</sub>-controller, Simulink model

- Which controller performed better in comparison with PI<sub>1</sub>-controller?

## References

Aström, K. and Hägglund, T. (2005). *Advanced PID control*. In *ISA-The Instrumentation, Systems, and Automation Society*. Research Triangle Park, NC.

Normey-Rico, J.E., Bordons, C., and Camacho, E.F. (1997). Improving the robustness of dead-time compensating pi controllers. *Control Engineering Practice*, 5, 801–810.

Normey-Rico, J.E. and Camacho, E.F. (2009). Unified approach for robust dead-time compensator design. *J. Process Control*, 19, 38–47.

Normey-Rico, J.E., Guzman, J., Dormido, S., Berenguel, M., and Camacho, E.F. (2009). An unified approach for dtc design using interactive tools. *Control Engineering Practice*, 17, 1234–1244.

Visioli, A. (2006). *Practical PID Control*. Springer, London.

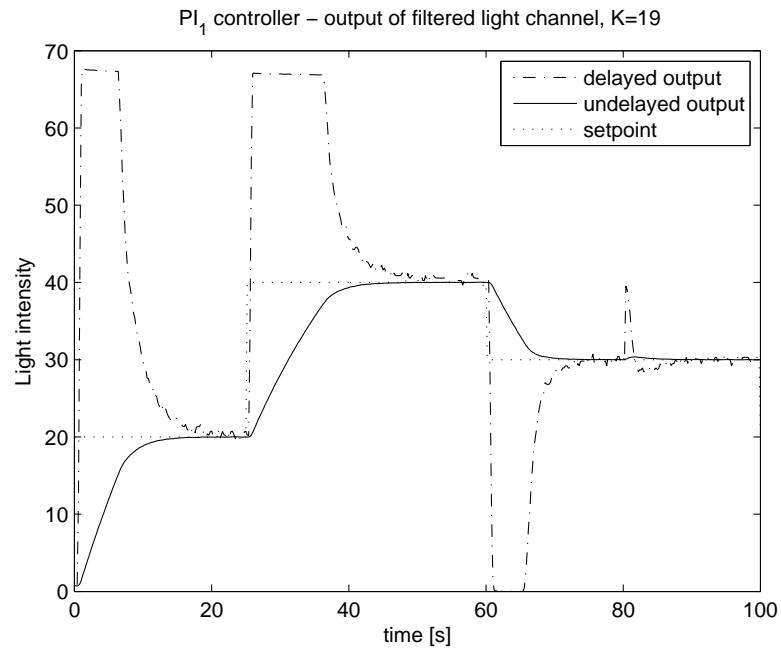


Fig. 9.47 PI<sub>1</sub>-controller, filtered light channel control for  $K = 19$

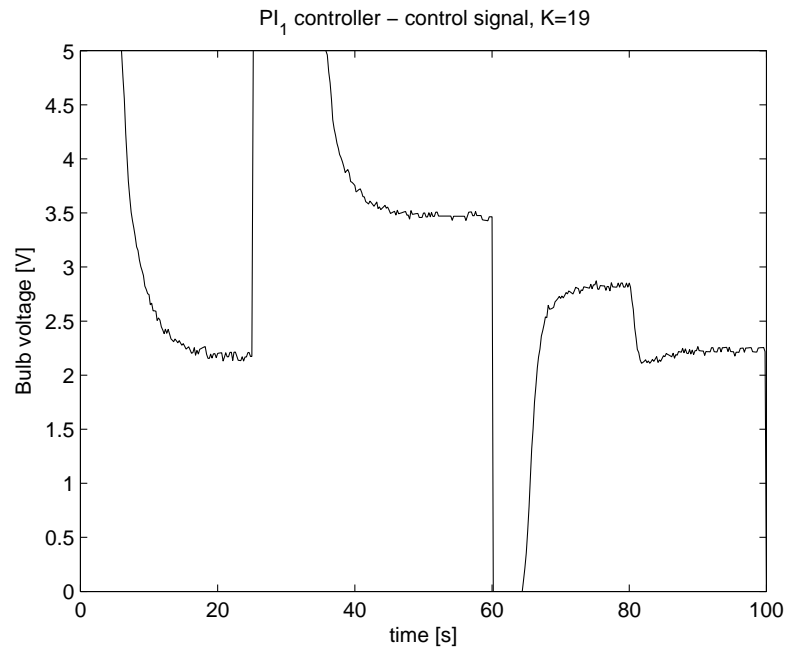


Fig. 9.48 PI<sub>1</sub>-controller, filtered light channel control for  $K = 19$



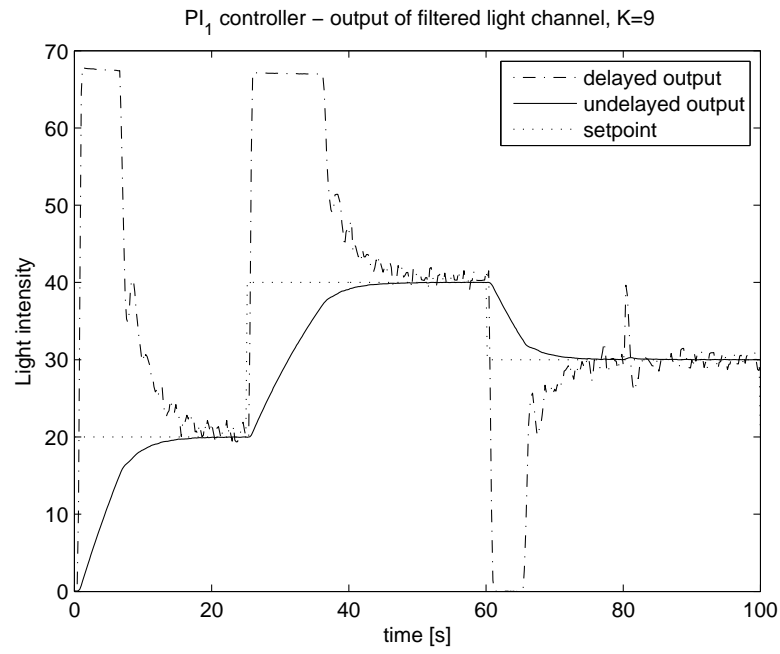


Fig. 9.49 PI<sub>1</sub>-controller, filtered light channel control for  $K = 9$

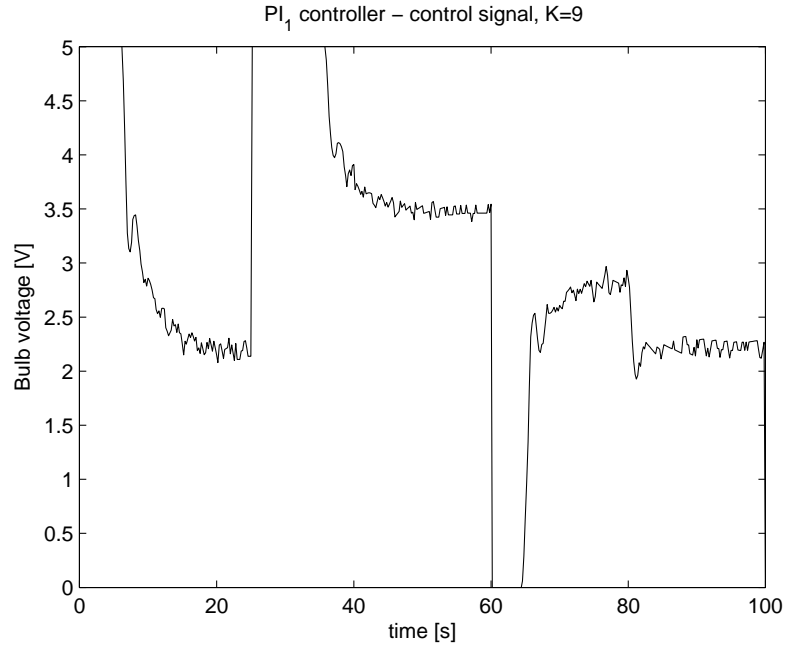


Fig. 9.50 PI<sub>1</sub>-controller, filtered light channel control for  $K = 9$

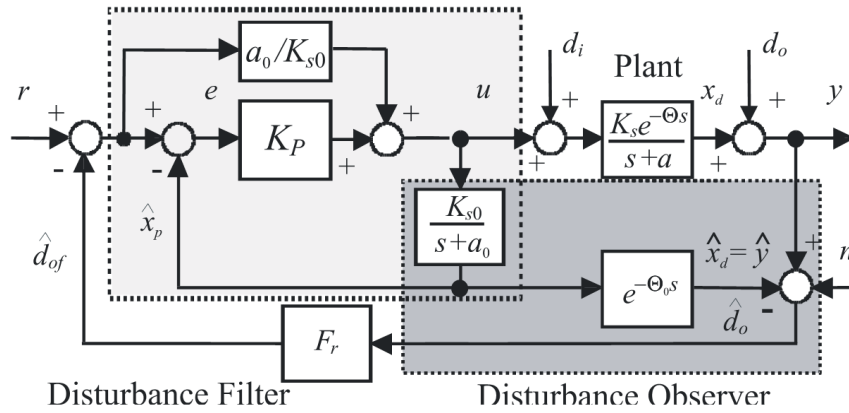


Fig. 9.51 Modified P-FSP with the primary loop using 2DOF P-controller with the disturbance filters

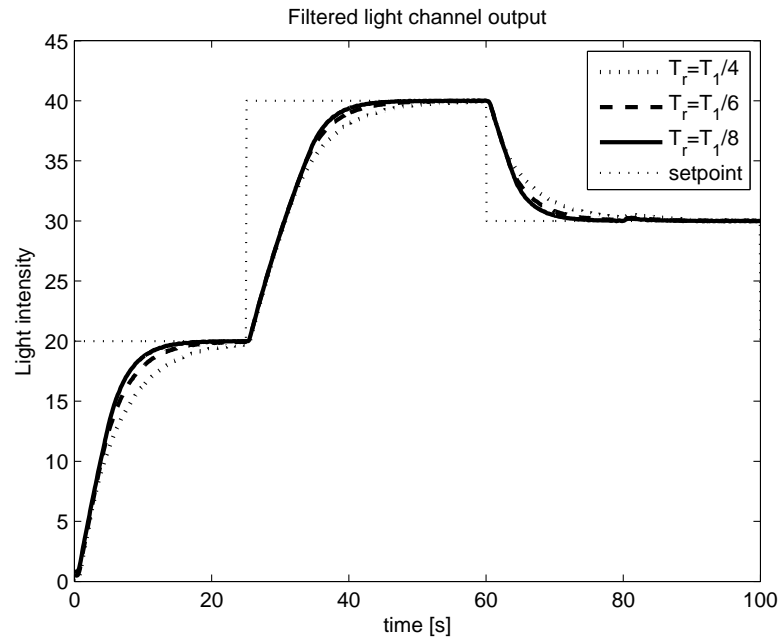


Fig. 9.52 FSP for filtered optical channel – system output

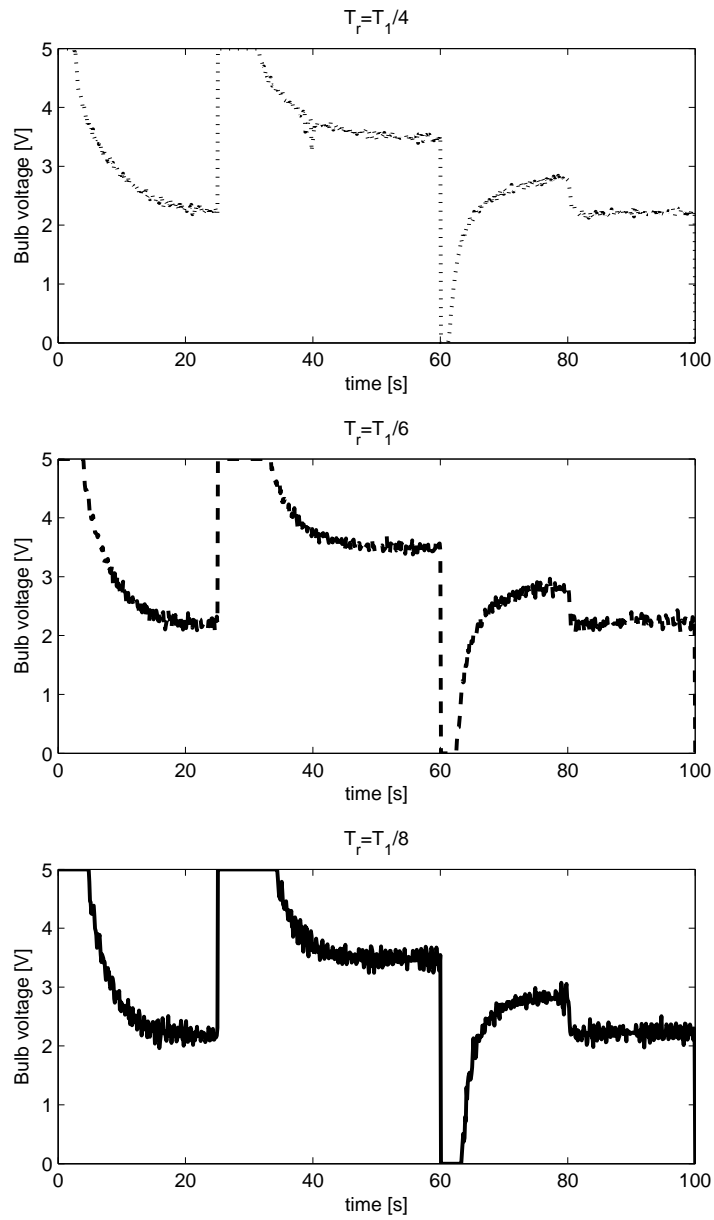


Fig. 9.53 FSP for filtered optical channel – control signal

*Comments – Remarks*

*Comments – Remarks*

*Comments – Remarks*

*Comments – Remarks*



